CFD-DEM Coupling for Systems of Fluid and Non-Spherical Particles

by

Davide Fantin

to obtain the degree of Master of Science in

Applied Mathematics

at the Delft University of Technology,

to be defended publicly on Friday November 9, 2018 at 13:30.

Student number:	4746880	
Project duration:	November 1, 2017 – October 31, 2018	
Thesis committee:	Prof. dr. ir. C. Vuik,	TU Delft, supervisor
	Prof. dr. ir. A. W. Heemink,	TU Delft
	Dr. ir. F. Bos,	Dynaflow Research Group

TU Delft

Preface

This Thesis describes and presents the MSc project performed in the framework of MSc *Computer Simulation for Science and Engineering* at TU Delft and TU Berlin.

The research was carried out at Dynaflow Research Group. I would like to thank Dr. ir. Frank Bos for the opportunity to work side by side with experienced engineers, researchers and developers. His interest in the project and his ideas for new directions and implementations were trilling challenges that kept the fire of the project burning. I would like to thank all the people that worked around me for their professionalism and their constant willingness to sharing ideas and feedback on the research. Sometimes, interesting talks and discussions on technical topics prevented us to go back home at reasonable time.

I would like to thank Prof. dr. ir Kees Vuik (TU Delft) for the constant supervision and careful reading of all my intermediate reports. His help was fundamental, especially in the critical phases of the project. I always found support in our discussions and I am thankful for his positive attitude when commenting satisfactory results as well as aspects to improve.

Not only am I grateful to Prof. Kees Vuik for his supervision, but also for his constant work with Prof. Reinhard Nabben (TU Berlin) in the organization of the Double Degree MSc programme in *Computer Simulation for Science and Engineering*. Thanks to their vision and their efforts, I had the opportunity to attend courses, lessons and seminars from two European high-level universities, obtaining two degrees (MSc Applied Mathematics at TU Delft and MSc Scientific Computing at TU Berlin) in one effectively structured program. Living abroad enlarging the personal horizons, exploring cultures, embracing the German and the Dutch lifestyles, meeting people from diverse contexts are the most astonishing experiences that I have lived so far. This was possible only because of their constant work in the organization of the course of study in a flexible but effective way.

My deepest gratitude goes to my family. I can honestly say that my achievements could not be obtained without their full support, in the happiest but especially in the darkest moments. I will never thank them enough for their help and also for their attempts to understand my needs to study and work a little bit farther from home than they expected. I am thankful for the freedom that they always gave me, their unconditional support and trust.

Finally, I would like to sincerely thank my friends: regardless of the distance, with long talks or just a few words, they supported me while dealing with the difficulties of life, supporting me in the tough moments and sharing with me the shining ones.

I like to consider this Thesis as a material realization of a process that started years ago, when I put my first foot in a school building. I would never expect to live all these extraordinary experiences and to have the opportunity to embrace all these exciting challenges. Hence, my last thank goes to the would-never-expect things in life and the astral conjunctions that align apparently random points in marvellous paths.

Davide Fantin
Delft, November 2018

Contents

Introduction

Mathematical models and their simulations are becoming more and more fundamental in both industrial and academical environments. Thanks to the increasing computational capabilities, it is nowadays possible to simulate industrial and physical processes, exploring their evolution in time without performing them in real-life.

The field of numerical simulations has an extremely wide set of possible applications. Engineering problems can be faced with the help of computer simulations, which can provide fast and reliable results in complex situations. Design products optimizing specific physical properties, exploring scenarios, predicting vulnerabilities are just a few of the cases in which a numerical approach can be efficient and low-cost. Applications of simulations have been developed in a huge variety of fields, both in fundamental academic research and in industrial applications, e.g. petrochemical, marine, pharmaceutical and aeronautical industries. The main feature that numerical simulations provide is optimal and low-cost development of products and services. To this purpose, simulations of fluid/fluid, fluid/solid particles systems are essential tools and require deep research to obtain realistic results.

Using a "mathematical" perspective, it is possible to claim that *at a first order approximation* the entire world is based on fluids/particles interactions.
This is the reason why one of the most challenging fields is the study of fluid/particles systems, which has applications in uncountable different contexts. Due to the extreme importance of simulations of fluid-particle systems, in the last decades academical and industrial researches focused more and more on the attempt to develop efficient numerical approaches to be used in all kinds of applications. Numerical methods and their implementations are continuously improved and the literature on mathematical modeling and numerical simulations has become extensive. Even if progress has grown exponentially over the years, a lot of work is still to be done, both in the modeling and in the simulation.

In fluid-particle systems, the fluid behavior is described by fluid dynamics, more precisely by Navier-Stokes equations and in this work it is simulated through methods of Computational Fluid Dynamics (CFD), whereas the dynamics of discrete solids or particles is described using Newton,s law on each particle and, in this work, simulated using Discrete Element Method (DEM).

The aim of this work is to improve coupling capabilities between an open source CFD toolbox (OpenFOAM) and DEM. The current implementation in OpenFOAM already includes a rudimentary interaction between the modeled particles themselves and between the particles and flow, but it has significant limitations on the size and shape of the modeled particles. In fact, particles are only approximated as spheres, but the most problematic aspect of the implementation is the ratio between particles and the fluid grid. An efficient framework developed recently is CFDEM, which couples OpenFOAM and LIGGGHTS, a particle solver. We will use this architecture to couple OpenFOAM and HADES, a particle solver developed by Dynaflow Research Group. Afterwards, we will extend the capabilities of the software implementing a multisphere approach to handle non-spherical particles and so generalize the method to situation closer to reality.

We divided the project report in three parts: Theoretical Knowledge, Implementation and Applications.

In part I, we give an introduction to the field of fluid-particle simulation. A literature study has been carried out to develop the necessary basis to understand the state-of-the-art approaches adopted to simulate fluid/particle systems. The literature study was performed in the first three months of the project duration.

Firstly, in Chapter 1 the equations of incompressible fluid dynamics are presented and derived. A brief introduction to the open source software OpenFOAM is given. Then, in Chapter 2 the Discrete Element Method for modeling particle/particle interactions is described and two software packages for DEM simulation are presented: HADES and LIGGGHTS, respectively. In Chapter 3 the state-of-the-art techniques for the coupling between CFD and DEM are described in detail. Finally, in the last Chapter of the part, some research questions are stated. The successive part of the Thesis will deal with the proposal of solutions to these questions.

The main part of this Thesis report is part II, where we describe new implementations and improvements of the current software features. In particular, in Chapter 5 we describe in detail the CFDEM framework and we substitute the particle solver (from LIGGGHTS to HADES) describing the infrastructure developed for the purpose. Afterwards, in Chapter 6 we describe some improvements of the coupling code, managing to not restart the particle solver at each iteration and allowing CFD parallel computations. In Chapter 7 we give an overview of the possible strategies to generalize the code for non-spherical objects, we select the multisphere approach and we implement in HADES, extending the particle solver features. Two tests to verify the new software capabilities are performed and discussed in Chapter 8.

Finally, in part III, we propose in Section 8 two tests to validate the newly developed features and in Section 9 two possible applications of the new features and we discuss their accuracy in solving academic and industrial sample cases.

The last Chapter is dedicated to a summary of the results of the project and to state possible directions for future researches and developments.

Theoretical Foundation

1 Incompressible Fluid Dynamics

In this chapter fluid flows are considered. We briefly introduce the equations that arise from incompressible fluid dynamics. The Navier-Stokes equations are derived under the hypothesis of incompressibility. Then, OpenFOAM, an open source software package for fluid simulations, is introduced.

1.1. Derivation of Incompressible Navier-Stokes equations

Derivation of Navier Stokes equations appears in every monograph dedicated to fluid and flows, therefore in this report only main ideas are presented. We follow the derivation of the equation in [28]

The first step in the derivation of Navier Stokes equations is to apply the principle of mass conservation to a control volume $V(t)$ that contains a specific collection of fluid particles. Let $\rho(\mathbf{x}, t)$ be the density of the fluid at point $\mathbf{x}$ at time t, then conservation of mass applied on control volume $V(t)$ reads:

$$\frac{d}{dt}\int_{V(t)} \rho(\mathbf{x}, t)\, dV = 0. \tag{1.1}$$

Let $\mathbf{u}(\mathbf{x}, t)$ be the velocity field, $A(t)$ the surface of the control volume and n the outward normal to the surface. Then applying Reynolds transport theorem we get:

$$\int_{V(t)} \frac{\partial}{\partial t}\rho(\mathbf{x}, t)\, dV + \int_{A(t)} \rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t)\cdot \mathbf{n}\, dA = 0. \tag{1.2}$$

Applying divergence theorem to the surface integral we get:

$$\int_{V(t)} \left\{ \frac{\partial}{\partial t}\rho(\mathbf{x}, t) + \nabla\cdot(\rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t)) \right\} dV = 0. \tag{1.3}$$

Equation (1.3) has to be valid for all possible control volumes, in particular for vanishing control volumes. Therefore, the integrand in the left hand side has to be zero. We derived the differential form of the mass conservation, or the *continuity equation*.

$$\frac{\partial}{\partial t}\rho(\mathbf{x}, t) + \nabla\cdot(\rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t)) = 0. \tag{1.4}$$

Usually, the continuity equation (1.4) is written as:

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0. \tag{1.5}$$

Adding the hypothesis of incompressible flows, equation (1.5) becomes:

$$\nabla\cdot\mathbf{u} = 0. \tag{1.6}$$

We will follow the same procedure applying the principle of momentum conservation to the control volume $V(t)$. The conservation of momentum deals with forces applied to the control volume. We will divide the forces in two contributes: *body forces*, that act without physical contact with the element and *surface forces*, that act through direct contact with the surface of the element. To this purpose, let $\mathbf{f}(\mathbf{x},t)$ be the body force per unit mass on the fluid inside $V(t)$ and let $\mathbf{t}(\mathbf{n},\mathbf{x},t)$ be the surface force per unit area on the surface $A(t)$, usually called *stress vector*.

$$\frac{d}{dt}\int_{V(t)}\rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t)dV=\int_{V(t)}\rho(\mathbf{x},t)\mathbf{f}(\mathbf{x},t)dV+\int_{A(t)}\mathbf{t}(\mathbf{n},\mathbf{x},t)dA \tag{1.7}$$

We use the assumption on curvature of Cauchy, i.e. $\mathbf{t}(\mathbf{n},\mathbf{x},t)=\mathbf{T}(\mathbf{x},t)\mathbf{n}$, where $\mathbf{T}(\mathbf{x},t)$ is usually called *stress tensor*. This corresponds to the hypothesis that the stress vector is a linear function of the stress tensor and the normal derivative to the surface. Applying the Reynolds Transport Theorem and the divergence theorem on the surface integral we get:

$$\int_{V(t)}\left\{\frac{\partial}{\partial t}\rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t)+\nabla\cdot(\rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t))-\rho(\mathbf{x},t)\mathbf{f}(\mathbf{x},t)-\nabla\cdot(\mathbf{T}(\mathbf{x},t))\right\}dV=0. \tag{1.8}$$

As the case of mass conservation, equation (1.8) has to hold for vanishing control volumes, therefore the integrand on the left hand size has to be zero.

$$\frac{\partial}{\partial t}\rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t)+\nabla\cdot(\rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t))-\rho(\mathbf{x},t)\mathbf{f}(\mathbf{x},t)-\nabla\cdot(\mathbf{T}(\mathbf{x},t))=0. \tag{1.9}$$

Using the continuity equation (1.4) and the shorter notation we can simplify the equation (1.9) and we obtain the *Cauchy equation of motion*:

$$\rho\left(\frac{\partial\mathbf{u}}{\partial t}+\mathbf{u}\cdot\nabla\mathbf{u}\right)=\nabla\cdot\mathbf{T}+\mathbf{f}. \tag{1.10}$$

Equation (1.10) is able to describe the conservation of momentum at differential level for both fluids and solids. In order to describe fluids we have to consider a constitutive equation for the stress tensor $\mathbf{T}$. We consider *newtonian* fluids, in which the stress tensor $\mathbf{T}$ is a linear function of the rate of strain tensor $\mathbf{E}=\frac{1}{2}(\nabla\mathbf{u}+\nabla\mathbf{u})^T$ and we add the *incompressibility* assumption. Hence, we get the following equation for the stress tensor $\mathbf{T}$:

$$\mathbf{T}=-p\mathbf{I}+2\mu\mathbf{E}, \tag{1.11}$$

where p is the pressure and μ is the viscosity of the fluid. Substituting the constitutive relation on the Cauchy equation of motion (1.10) gives:

$$\rho\left(\frac{\partial\mathbf{u}}{\partial t}+\mathbf{u}\cdot\nabla\mathbf{u}\right)=-\nabla p+\mu\Delta\mathbf{u}+\mathbf{f}. \tag{1.12}$$

Considering the continuity equation (1.6) and the Cauchy equation for incompressible newtonian fluids (1.12) we finally get the *Navier Stokes equations*:

$$\begin{cases}\nabla\cdot\mathbf{u}=0\\ \rho\left(\dfrac{\partial\mathbf{u}}{\partial t}+\mathbf{u}\cdot\nabla\mathbf{u}\right)=-\nabla p+\mu\Delta\mathbf{u}+\mathbf{f}.\end{cases} \tag{1.13}$$

1.2. OpenFOAM

The Navier Stokes equations derived in the previous section are a system of non-linear partial differential equations, which manage to describe the behavior of general 3D flows, laminar or turbulent. They are extremely efficient in modeling all kind of flows but, due to non-linearities, an analytic solution is not available for the general case. Therefore, up to now, only numerical solutions are available. Numerical solutions are determined by discretization of the equations and on development of algorithms to get an approximation of the solution of the original (continuous) problem.

Several commercial software packages are able to simulate efficiently the behavior of fluids in several different situations. In this report we will consider an open source software, OpenFOAM and we will refer to its user's guide [14]. OpenFOAM (*"Open source Field Operation And Manipulation"*) is a C++ toolbox for the

development of customized numerical solvers for the solution of continuum mechanics problems. It also provides pre- and post-processing utilities.
Basically, OpenFOAM is a library, which can be used to build the so called *applications*. Applications can be *solvers* or *utilities*. Solvers perform the calculation to solve a specific problem. Utilities prepare the mesh, set-up the simulation case and process the results.

OpenFOAM uses Finite Volume Method (FVM) for the discretization and the solution of partial differential equations. The idea is that the domain is divided in control volumes. On each control volume, partial differential equations are discretized and solved. A dissertation of the FVM is beyond the scope of this work. Details of the implementation of the method and of the various techniques developed over the years can be found in monographs like [45] or [13].

OpenFOAM solvers have been developed for a broad set of problems. Some areas in which standard solvers available for fluid mechanics are: potential flows, incompressible/compressible flows with DNS, RANS and LES capabilities, multiphase flows and particle-tracking solvers. Other fields in which OpenFOAM has been used are: combustion, conjugate heat transfer, molecular dynamics, electromagnetism and solid dynamics.

Main built-in solvers

Several solvers have been developed for different applications. Here we list the most relevant to the purpose of this project:

Incompressible Flows:

- `icoFoam` Transient solver for incompressible, laminar flow of Newtonian fluids.
- `simpleFoam` Steady-state solver for incompressible, turbulent flow, using the SIMPLE algorithm.
- `pisoFoam` Transient solver for incompressible, turbulent flow, using the PISO algorithm.
- `pimpleFoam` Large time-step transient solver for incompressible, turbulent flow, using the PIMPLE (merged PISO-SIMPLE) algorithm.
- `pimpleDyMFoam` Transient solver for incompressible, turbulent flow of Newtonian fluids on a moving mesh, with possibility of local refinements.

Multiphase flows:

- `interFoam` Solver for 2 incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach.
- `multiphaseInterFoam` Solver for n incompressible fluids which captures the interfaces and includes surface-tension and contact-angle effects for each phase.

Particle-tracking flows

- `DPMFoam` Transient solver for the coupled transport of a single kinematic particle cloud including the effect of the volume fraction of particles on the continuous phase.
  A more complete discussion of this solver is provided in section 3.6
- `MPPICFoam` Transient solver for the coupled transport of a single kinematic particle cloud including the effect of the volume fraction of particles on the continuous phase. Multi-Phase Particle In Cell (MPPIC) modeling is used to represent collisions without resolving particle-particle interactions. Collisions are in fact represented by models which evaluate mean values calculated on the Eulerian mesh. A severe limitation of the solver is that the size of particles must be small compared to the Eulerian grid for accurate interpolation. This coupling will be defined as unresolved coupling in the following chapter. The solver provides reliable results in dense particle flows (more than 5% by volume), but since it does not resolve particle-particle interactions it is not useful for the aim of this work.

How to use the built-in solvers
Each simulation takes place in a specific directory, created by the user. In this directory there have to be three subdirectories:

- `system`
  It contains files to control the generation of the mesh and the integration method used to solve the specific problem.
- `constant`
  It contains the specification of constant of the problem and it store the mesh, once it has been generated following the instructions contained in the system directory
- `0`
  It contains one file per variable of the problem. Each file contains initial and boundary conditions for the specific variable.

All the files required by the simulation are text files with an appropriate syntax. Usually, two commands are necessary to perform a simulation: `blockMesh` generates the mesh and then a solver is used to actually perform the calculation, i.e. if we want to use icoFoam solver, it is enough to use the command `icoFoam`. Other commands may be required in specific cases.

Once the simulation has been performed, other directories will be created in the directory of the project: OpenFOAM will store the evolution of the solution in time with an user-defined time step, usually larger than the actual time step used in the calculations. Each directory will contain one file per variable, and in that file the numerical values for that variable are stored. Usually the visualization of the results is done via ParaView, an open source multiple-platform application for interactive, scientific visualization.

2

Particle/Particle interactions

In this Chapter, the interaction between particles in pure granular flows is considered.

Two different approaches are possible to describe particle/particle interaction: Lagrangian tracking or Eulerian modeling approaches.

- Lagrangian approach: individual particles (or parcels of) are tracked through the field and properties of each particle are evaluated.
  examples: Discrete Element Method (DEM), Discrete Parcle Method (DPM)
- Eulerian approach: sets of algebraic conservation equations are solved simultaneously for each node in the field.
  examples: Two Flow Model (TFM)

In this work, we focus on the description of Discrete Element Method. Firstly, we introduce the method. Then, two open source software packages, HADES and LIGGGHTS, that apply DEM are introduced and presented.

General overview of DEM

DEM is based on the assumption that the material in consideration is made of separate, discrete particles. Granular matter, bulk material, solutions, liquids, powder and rocks are the most common example of application of DEM.
The first steps of development of a DEM for a particles system are the generation of a model, the orientation in space of all the particles and assignment of an initial velocity to them. The forces applied on each particle are computed from the initial data and depend on the model used to describe contact between particles and on the physical laws relevant on the specific problem. Possible important contributions can be given by:

- Macroscopic: Friction, Contact plasticity, Attractive potentials (cohesions, adhesion), Gravity
- Microscopic: Electrostatic attraction (Coulomb), intra-molecular forces (Van der Waals)

Every force taken into consideration for a specific problem is summed to have the total force exerted on every particle.

Once the total force acting on each particle is computed, it is possible to perform an integration in time to evaluate the new positions and the velocities of the particles, using suitable integration method, as the Verlet algorithm or symplectic integrators. We will analyze numerical methods suitable to this purpose in Chapter 4.

The simulation consists in applying this steps until a suitable final time is reached. Not all the possible forces taken into account have the same computational cost. A peculiar case is given by long-range forces, which require to evaluate the interaction between each pair of particles. In this case, the computational cost of the method increases quadratically with the number of particles considered. Ad-hoc methods are developed to

reduce the computational effort, for example combining particles that are far from the particle in consideration and considering them as a single pseudo-particle.

The main disadvantage of DEM is that the maximum number of particles is strongly limited to computational resources: real-life situations are often characterized by a huge demand in computational power. To reduce computational time, cluster of GPUs can be used.

Despite this problems, DEM is a powerful resource to simulate a wide variety of granular flows and rock mechanics problems and it can be the unique way to study micro dynamics of systems in which measurements are nearly impossible due to the small scale.

2.1. Discrete Element Method

The Discrete Element Method (DEM) is a Lagrangian method used for calculating the dynamics of large granular systems. In this presentation we use results presented in [11] and [18]. The particle flow is resolved at the particle level. In fact, as described above, DEM calculates the trajectory of each particle considering the influences by other particles, walls or other problem-specific forces. The motion of a particle consists of a rotational and a translational component, therefore the equations that describe the method are the Newton's laws for translations and rotations:

$$\begin{aligned} m_i \frac{du_i}{dt} &= F_i, \\ I_i \frac{d\omega_i}{dt} &= T_i, \end{aligned} \tag{2.1}$$

where m_i is the mass of the particle i, F_i is the force applied to the particle and u_i is the velocity, which is unknown; I_i is the inertia tensor, T_i is the torque applied to the particle and ω_i is the angular velocity, which is also an unknown. The force F_i has to be modeled in order to describe the particle/particle interactions. Usually, as described in [18], F_i takes into account:

- a gravitational component $m_i g$
- particle-particle collisions $\sum_{N_p} F_{i,p}$
- particle-wall interactions $\sum_{N_w} F_{i,w}$
- cohesive interactions $\sum_{N_p} F_{i,c}$

where N_p is the number of the particles in the system and N_w is the number of the walls. We have to mention that other problem-specific forces can be considered, like electromagnetic or chemical contributions. Using the aforementioned forces, the force F_i on the particle is given by:

$$F_i = m_i g + \sum_{N_p} F_{i,p} + \sum_{N_w} F_{i,w} + \sum_{N_p} F_{i,c} \tag{2.2}$$

In this case, Netwton's laws (2.1) take the form:

$$\begin{aligned} m_i \frac{du_i}{dt} &= m_i g + \sum_{N_p} F_{i,p} + \sum_{N_w} F_{i,w} + \sum_{N_p} F_{i,c} \\ I_i \frac{d\omega_i}{dt} &= T_i \end{aligned} \tag{2.3}$$

After having derived the equation (2.3) it is straightforward to notice that the next step is to model the forces $F_{i,p}$, $F_{i,w}$, i.e. the contributions of the interaction between particle i and all other particle and the walls, respectively, and and $F_{i,c}$, i.e. the contribution of the cohesive forces.

For the sake of simplicity, in the development of DEM each particle is assumed to be a sphere. This will impose a limitation on the accuracy of the method, but non-spherical particle can be approximated by several spheres glued together. The assumption is a necessary simplification, since it allows to develop easily contact and cohesive models and to consider the arising torque T_i as generated exclusively by tangential component of the force F_i.

Particle-particle interactions

For the purpose of modeling the interactions that arise form particle-particle collisions, two different approaches can be used: hard sphere model and soft sphere model.

- *Hard sphere approach*
  The particles are impenetrable and the contacts are instantaneous and perfectly rigid. Only binary contacts are considered and long-distance particle forces are neglected. This approach is mainly useful in dilute systems, where the number of binary collisions prevail.
- *Soft sphere approach*
  When solids exert forces on each other, they are subjected to deformation. In this model, deformation is replaced with an overlap between the two particles taken in consideration and the arising force will depend on this overlap. The results are more accurate than the hard sphere model, but the computational effort is much bigger.

The Discrete Element Method is based on the soft sphere model.

The main assumption is that particles which are in direct contact with the particle in consideration influence its motion. A collision between elastic particles generates repulsive forces, which will be directly proportional to the deformation, described by the overlap. Since we are considering deformations, we have to notice that deformation of a body implies energy loss, which depends on deformation speed.

A very intuitive mechanical analogy to this process is given by the spring-damper system. In Figure 2.1 a visualization of such a system is provided. In a spring-damper system, the motion of a body with mass m is described by

$$m\ddot{x} + \eta\dot{x} + kx = 0, \tag{2.4}$$

where η is the damping coefficient of the dash-pot, k is the stiffness of the spring and x is the distance form the equilibrium position.

Using this linear ODE (2.4) not all details of the physical process are described, but we will adopt this model since it allows to capture fundamental properties of the system, for example the loss of kinetic energy. In fact, in the dash-pot model, loss of kinetic energy is modeled, which is in accordance to the hypothesis of energy loss due to deformation during the collision between particles. The solution of the linear ODE is parameter-dependent, and can be over-damped, critically damped or under-damped. Usually the collision of particle gives rise to under-damped solutions.

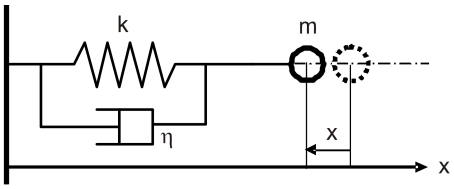


Figure 2.1: **Analogy particle collisions and spring-damper system**. The image has been taken from [11], page 125, for its simplicity.

The analogy between the two phenomena lies in the following processes. Due to external forces, one particle A is pushed towards another B. In the spring-damper system this corresponds to a force that moves the equilibrium position of x. When the point leaves the equilibrium, the spring compresses and a repulsive force is created. This repulsive force corresponds to the reaction force caused by B acting on the particle A. Due to this reaction force, particle A starts to return to its original position after having reached the maximum displacement. Due to energy loss, the velocity of returning will be lower than the velocity before collision.

The aim is to develop a model for the force $F_{i,p}$, which describes the force applied to particle i due to particle-particle interactions. For the sake of simplicity, we write F_i.
The force F_i will be the sum of all the collisions with the j particles that are in contact with particle i. Therefore we can write

$$F_i = \sum_j F_{ij}. \tag{2.5}$$

We define δ as the overlap that is developed when two particles collide. We consider a collision between particle i and particle j. The overlap δ will have both normal and tangential component δ_n,δ_t. The same applies to the force F_{ij}: we decompose it in normal component F_{nij} and in tangential component F_{tij}. Hence, equation (2.2) becomes:

$$F_i = \sum_j \left(F_{nij} + F_{tij}\right). \tag{2.6}$$

Different equations model these two contributes.
Now we use the analogy between particle collisions and spring-damped systems to build a model for normal and tangential component of the force applied to the particles. We will deal with the stiffness parameter k, the damping coefficient η and the friction coefficient f.

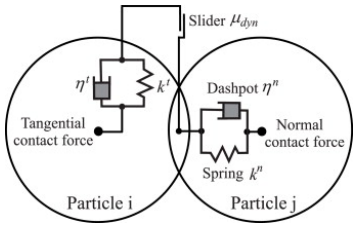


Figure 2.2: **Spring-damper system applied to contact between particles**. The image is taken from [2].

Firstly, we model the normal component of the force, i.e. F_{nij}.
The normal component is given by the sum of the forces due to the spring and the dash-pot. Using the Hertzian contact theory, the force is given by:

$$F_{nij} = (-k_n \delta_n^{3/2} - \eta_{nj}(\mathbf{v}_i - \mathbf{v}_j)\cdot\mathbf{n})\mathbf{n} \tag{2.7}$$

where k_n is the stiffness coefficient in the normal direction, η_{nj} is the damping coefficient in the normal direction, $\mathbf{v}_i$ is the velocity of particle i and $\mathbf{n}$ is the unit vector with direction the line that connects the centers of particles i and j. The power 3/2 of the displacement may seem strange, but the results are in good accordance with the experimental cases.

Now we model the tangential component of the force, F_{tij}.
For the tangential component we have to distinguish two cases, depending on the ability of the sphere to slide or not. We use a Coulomb-type friction law on the first case, whereas we develop an *ad-hoc* model for the second. Hence the force is given by:

$$F_{tij} = \begin{cases} -f|F_{nij}|\mathbf{t} & |F_{tij}| \geq f|F_{nij}| \\ -k_t\delta_t - \eta_{tj}\mathbf{V}_{ct} & \text{else} \end{cases} \tag{2.8}$$

where k_t is the stiffness coefficient in the tangential direction, η_{tj} is the damping coefficient in the tangential direction and f is the friction coefficient which is measured empirically, $\mathbf{t}$ is the unit vector in the direction of $\mathbf{V}_{ct}$, which is the slip velocity and is modeled as:

$$\mathbf{V}_{ct} = (\mathbf{v}_i - \mathbf{v}_j) - ((\mathbf{v}_i - \mathbf{v}_j)\cdot\mathbf{n}) + a_i\omega_i \times \mathbf{n} + a_j\omega_j \times \mathbf{n}$$

where a_i and a_j are the radii of the particles i and j.

We will now focus on the modeling for the stiffness coefficients k_n and k_t and the damping coefficients η_n and η_t.

We use Hertzian contact theory for the normal stiffness coefficient k_n. If we know Young's modulus E and Poisson ratios σ for the particles i and j we have:

$$k_n = \frac{4}{3}\left(\frac{1-\sigma_i^2}{E_i} + \frac{1-\sigma_j^2}{E_j}\right)^{-1}\left(\frac{a_i + a_j}{a_i a_j}\right)^{-1/2}. \tag{2.9}$$

We use Mindlin's theory for the tangential stiffness coefficient k_t. Knowing the shear modulus H of the two particles i and j we have:

$$k_t = 8\left(\frac{1-\sigma_i^2}{H_i} + \frac{1-\sigma_j^2}{H_j}\right)^{-1}\left(\frac{a_i + a_j}{a_i a_j}\right)^{-1/2}\delta_n^{1/2}. \tag{2.10}$$

In literature, Hertzian-Mindlin theory is sometimes substituted with Hooke Laws for the modeling of the elastic coefficient k_n and k_t. The usage of Hertzian or Hooke theory has become standard and it is commonly accepted.

Instead, for the damping coefficients a variety of models has been proposed and it is not rare to see the damping coefficient used as tuning parameters validate a particle solver. One of the first models developed, proposed by Cundall and Strack, used the critical parameter of the spring damper analogy as the damping coefficient of the contact force, since the bouncing motion after collision should be damped as soon as possible. In this outlook, the normal and tangential coefficients are modeled as:

$$\eta_n = 2\sqrt{mk_n}, \tag{2.11}$$

$$\eta_t = 2\sqrt{mk_t}. \tag{2.12}$$

Other models consider the parameter as dependent to the derivative of the displacement, leading to a partial-non linear or a full non linear force model. An overview of the models can be found in [26] and [8]. Nevertheless, the same authors claim that often, in current implementations, the damping coefficient appear to depend to some user-input parameters, which are tuned to obtain simulations close to the expectation from a physical point of view.

In the software that we will take under consideration, i.e. HADES, which is introduced in Section 2.2, the model implemented is:

$$\eta_n = \texttt{NormCoeff}\ \frac{m_1 m_2}{m_1 + m_2}\delta_n^{1/2}, \tag{2.13}$$

$$\eta_t = \texttt{TangCoeff}\ \frac{m_1 m_2}{m_1 + m_2}\delta_n^{1/2}, \tag{2.14}$$

where `NormCoeff` and `TangCoeff` are input parameters and m_1, m_2 are the masses of the respective particles.

In conclusion, a lot is to be done in the modeling of damping coefficients to achieve reliable simulations of complex real-life situations and extensive further research will be necessary in future to compare simulation results and experimental data.

Cohesive Interactions

Granular materials have the ability to resist external tensile stress. This property is caused by microscopic attraction forces between particles, called cohesive interactions, which may have physical and chemical origins. For this part, we follow the dissertations [11] and [39].

The cohesive interaction have the effect to resist to separation, shear or rolling of two particles, restricting the relative particle displacements. Not all the granular materials manifest appreciable effects of the cohesive interactions: it is possible to classify the materials in *weakly cohesive* and *strongly cohesive*, depending on the influence of the cohesive forces in the macroscopic behavior. Several physical phenomena can be responsible for the arise of cohesive forces. We will give three examples: electrostatic and Van der Waals forces as *bulk forces* and capillary bridge as *contact force*.

We first introduce the electrostatic case. Electrostatic forces occur if at the particle surface the electrical charges of opposite signs are not balanced. The radial force that arise is this case, between particles i and j is:

$$F_{ij} = \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_r l^2}, \tag{2.15}$$

where q_i and q_j are the electrical charges of particles i and j, ε_0 and ε_r are the vacuum permittivity and relative permittivity of the medium, respectively and l is the distance between the particle centers. This force is attractive if q_i and q_j have opposite signs, otherwise it is repulsive.

The Van der Waals forces are inter-particle forces that cause adhesion between particles between each other and the walls. They are particularly strong when smooth surfaces are brought to contact. The Van der Waals force between two spherical particles is described by:

$$F_{ij} = \frac{A}{12l^2}\frac{D_i D_j}{D_i + D_j}, \tag{2.16}$$

where D_i and D_j are the diameters of particles i and j respectively, A is a constant (*Hamaker Constant*) and l is the separation distance at the particles start to interact.

Cohesive forces can arise even when two particles are in contact, developing adhesion. An example is given by capillary bridges. The capillary force is given by:

$$F_{ij} = -2\pi R^*_{ij}\gamma\cos\theta, \tag{2.17}$$

where γ is the liquid surface tension and θ the contact angle and R^*_{ij} is the average of the radii of particles i and j.

Total Interaction

Once we have modeled each component of the forces acting on the particle i, we can go back to (2.3) and rewrite it in a simpler way. We do not consider all the particles, but only those that are in contact with particle i. Hence we do not sum over N_p but over j. Moreover, we sum all the different contributions in the normal direction in F^{tot}_{nij} and in the tangential direction in F^{tot}_{tij}. Therefore we obtain:

$$\begin{aligned} m_i\frac{du_i}{dt} &= m_i g + \sum_j \left(F^{tot}_{nij} + F^{tot}_{tij}\right) \\ I_i\frac{d\omega_i}{dt} &= \sum_j \left(a\mathbf{n}\times F^{tot}_{tij}\right) \end{aligned} \tag{2.18}$$

since the torque T_i is generated exclusively by the tangential component of the forces, as explained in the previous paragraph.

2.2. HADES

HAbanera's Discrete Element Simulator, named HADES, is a discrete element software package, developed by Habanera, that simulates granular flow or mixture problems.
As described in the previous section, using DEM the behavior of the entire material/mixture is simulated by considering contribution of each constituent in the mixture individually. In fact, performing a simulation of the complex interactions between the grains mutually and the influence of the environment on the individual grains, the behavior of the whole mixture can be evaluated.

In HADES the dynamics of the individual particles is evaluated by integrating Newton's second law of motion. With the technique adopted in the previous section, it is possible to model the total net forces and torques acting on a particle. This is obtained summing the individual forces F_{ij} and torques T_{ij} that act on body i over the number of actuators j, where as actuators we define the processes responsible for the arise of a force. Knowing the particle state (its position and its velocity) at a particular time, it is possible to obtain the state of particles at a later time integrating the above equations in time. To this purpose various numerical integration schemes can be used. Currently, HADES only supports explicit schemes.

Actuators and Models

Very different processes can be at the origin of forces and each one of them requires careful modeling. As stated before, we call each of these processes an actuator. The result of an active actuator on a particle is a force. For example, gravity, drag and contact are actuators that may contribute to the total net force that acts on a particle. In DEM, the force F_{ij} that acts on a particle i due to an actuator j is independent from the force F_{ik} that acts on the same body but from a different actuator k. Therefore each actuator can evaluate its influence on a group of particles, independently from any other actuators that may be active. For example, the evaluation of the gravity force can be performed independent from the evaluation of the drag force and contact force.

In HADES these actuators are encapsulated in so called Models. Each Model calculates its contribution to the total force that acts on each particle. The Models available for the evaluations of forces are *collision models*,

that calculate the inter-particle contact forces, *drag force models* that calculate the force on a body that moves through a medium and the *gravity model* that calculates the gravitational forces that act on a body.

Other implemented Models control the number of particles that are active during a simulation. To this purpose, a sink model and various generator model are available. The *sink model* deletes from the simulation the particles that enter a user-defined geometrical region, instead other *generator models* are able to generate particles of a user-defined shape and size, at user-defined locations at user-defined times during the simulation.

These Models can be added via the input file. The user specifies, in an ASCII input file, which models are active during the simulation and what values should be used for the relevant model parameters. For example, if gravity plays no role in the experiment that is being simulated, the user simply does not add this Model. This modular design gives HADES a very strong flexibility and since most of the features are encapsulated in independent Models, it also allows HADES to be extended in a simple way.

Physical objects, such as containers, hoppers, borders and particles, may have arbitrary shapes, but the evaluation of the mutual interactions between arbitrarily shaped objects is in general too computationally expensive. Hence, optimized algorithms are provided for simple shapes, like spheres and planes.

Property files

In order to run an HADES simulations, user-defined input needs to be considered. The user gives directives through the *property files*. The property file is a text file containing a set of name-value pairs, called properties, that describe the runtime parameters and the models to be used. The general syntax of a property is:

```
name = value;
```

The value of a property can also be a property set, so that it is possible to create a tree-like structure of properties. An example of same parts of a property file is given by the following script:

```
integrator =       // define integration scheme and configure integrator
{
    type       = "Verlet";
    ...
};

generator =       // define generator type and configure this generator
{
    type       = "ellipsoidGenerator";
    ...

generation =
    {
        nrOfBatches        = 1;
        bodiesPerBatch     = 8000;
        fireTime           = 0.0;
    };
};
```

Using this kind of syntax for the appropriate properties, the user is able to generate particles with particular shapes, choose which actuators to apply, define the time integration method and all the parameters required for the simulation.

We now list the built-in models, already available in the current version of HADES.

Built-in Models

Models to add bodies to a simulation:

- `ellipseGenerator` generates bodies of elliptical shape
- `ellipsoidGenerator` generates bodies of ellipsoidal shape
- `fromFileGenerator` generates bodies of which the shape description is obtained from file

Model that remove bodies from a simulation:

- `sink` removes bodies from the simulation

Models that apply contact forces and gravity to particles:

- `HertzContact` Hertz Mindlin contact between particles of spherical shape
- `HertzContactPlaneSphere` Hertz Mindlin contact between particles of spherical shape and infinite planes
- `gravity` calculates the gravitational force on bodies
- `segmentContact` contact model between particles of arbitrary shape

Other modulus provide a first (one-way) coupling between particles and fluid. for example the calculation of drag force on particles, but we will discuss them later in the section dedicated to the coupling.

Up to now, HADES Hertzian Contact Models for contact forces, but cohesive force are missing.

We now give two examples of the usage of HADES software. In Figure 2.3 the fall of particles in a 2D box is simulated. After the flow is stopped, a circular motion is imposed on a particle and the behavior of all the other particles of the system is explored. In Figure 2.4 the collision of 3D particles is explored. The particle in the right is provided with a velocity in left direction. All the collisions are accurately caught and in the final time of the simulation all the particles have a non-zero velocity in the left direction. Due to the lack of other collisions, the particle in the left moves away from the system.

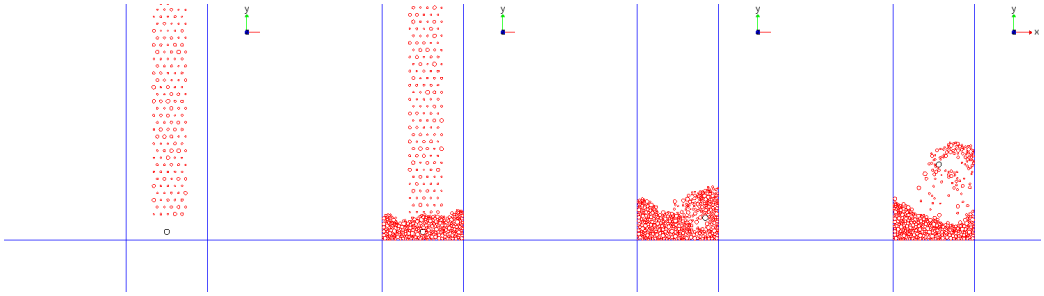


Figure 2.3: **HADES: 2D Particles fall and successive perturbation**

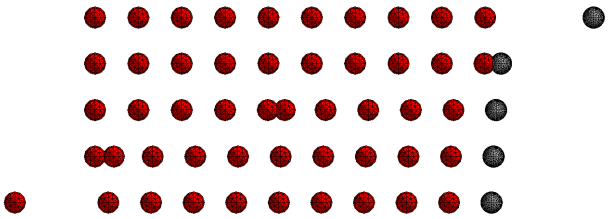


Figure 2.4: **HADES: 3D Collisions**

2.3. LIGGGHTS

LIGGGHTS (LAMMPS Improved for General Granular and Granular Heat Transfer simulations) is an open source DEM Particle Simulation Software. LAMMPS is a classical molecular dynamics simulator. It is widely used in the field of Molecular Dynamics. To perform DEM calculations, LAMMPS offers a GRANULAR package to perform these kind of simulations. LIGGGHTS aims to improve those capabilities with the goal to apply it to industrial applications. A very detailed documentation is provided in [10].

LIGGGHTS applies the Discrete Element Method described in the previous sections for simulations of interactions of particles between each other and with walls of containers. The equation of motion is numerically integrated, updating the state of each particle every time step. LIGGGHTS is a software written in C++ and it supports parallel computing via MPI. Mainly, it is open-source, but some features are provided only through a commercial license.

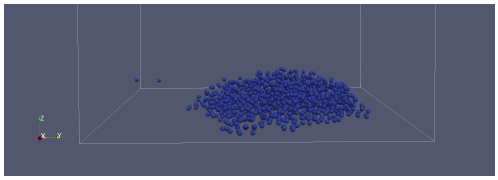
The user has to provide the set up of the simulation through an input text file, which contains geometrical and material properties, as well as the other parameters necessary for the simulation. It is possible to set the geometry of the problem directly on the input file or importing an user-defined mesh via a STL file. Then, the user runs an executable using the text file as input.

Input File

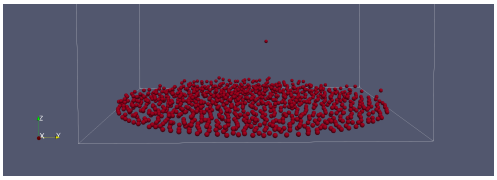
In the input file, the user has to set up the domain of the simulation, to choose the models to use (contact, cohesion, ...) and to fix all the parameters necessary for the simulation. The structure is usually given by:

- Definition of the shape of the particle and the boundaries of the domain
- Definition of the minimum distance for detecting the neighbors of the particles
- Setting of the physical parameters required by the model used
- Fixing the walls of the simulation or importing STL geometries
- Fixing the region of insertion of the particles
- Choosing output settings and run up to a certain final time

In LIGGGHTS it is possible to apply also cohesive models. A simulation of the drop of particles with and without cohesive forces has been performed and in Figure 2.5a and 2.5b final results are given. As expected, the case with cohesive forces results in the creation of a clump, whereas without cohesive forces a more ordered structure is obtained.



(a) **LIGGGHTS: Drop with cohesion**.



(b) **LIGGGHTS: Drop without cohesion**.

2.4. HADES vs LIGGGHTS

In order to have a complete overview of the software capabilities, it may be interesting to highlight the differences between the features implemented in HADES and in LIGGGHTS. Generally, LIGGGHTS is more complete than HADES, allowing simulations in more complex scenarios, but HADES possess a vary peculiar implementation of the Verlet algorithm, which is extremely interesting for the performances of the simulation. In LIGGGHTS, some contact models are available and it can handle complex geometries from STL files, whereas in HADES only the Hertzian contact model can be used and contacts can be detected only between spheres and between a sphere and an infinite plane. Hence, the capabilities of HADES are very limited from a geometrical point of view and for the models available.

Moreover, LIGGGHTS has support for parallel computations, allowing to achieve fast results in extremely complex scenarios, whereas the current implementation of HADES supports only serial computations, causing a severe limitation on the simulation of complex industrial situations.

Nevertheless, the key feature of HADES is to allow an integration in time with a non-fixed (a priori) DEM time step. In fact, the implementation uses a complex algorithm to tune the time step according to the values of the forces that act at each time step. If, during an integration step, a particle is subjected to a too-large displacement, the algorithm reduces the time step and tries again to fulfill the requirement. This feature is extremely important since it allows to decrease drastically the computation time. The user can intervene on the integrator using a safety factor (0,1) that multiplies the time step selected by the algorithm. This implementation is innovative and it is the reason why we will work on the coupling between OpenFOAM and HADES.

3

Coupling

Physical simulations often deal with interactions between solid particles and fluids, whereas up to now we considered these two phenomena as independent. In this Section, some methods for the interaction between fluids and particles are described and analyzed. This process is called *Coupling*, and we will focus on the coupling between CFD for fluid simulations and DEM for particle interactions.

The first classification of the coupling procedure regards the reciprocal influences between fluid and particles and particle on themselves. In fact, we distinguish 3 different cases:

- *1-way coupling*
  Fluid exerts influence on particle motion but not vice versa. Neglect particles interactions
- *2-way coupling*
  Fluid exerts influence on particle motion and vice versa. Neglect particles interactions
- *4-way coupling*
  Fluid exerts influence on particle motion and vice versa. Resolve particles interactions

Depending to the specific problem, the most efficient coupling can be implemented. The aim of this project is to study the 4-way coupling, therefore we will focus on this case.

In this Chapter, we firstly describe the modeling of fluid/particle interactions, describing the different contributions to the phase coupling. Then we discuss some general issues on the coupling procedure, i.e. the time step choice and the contact detection algorithm. Finally, we describe two different approaches that have been developed for the CFD-DEM interactions: *Resolved* and *Unresolved* couplings. Resolved coupling is useful when the size of the particle is bigger than the computational grid used for fluid simulation, whereas unresolved coupling deals with the case of particles which are smaller than the computational grid. We will describe in depth the details of this two distinct methods, for which we follow the references [18] and [25]. Some open source implementations of the coupling procedure are then listed and briefly described.

3.1. Modeling of Fluid/Particle Forces

The fluid dynamics force on a particle can be modeled as the superposition of different contributes. In this section we briefly describe the different components. It is important to notice that not every component will be relevant in every problem: the formulation of fluid/particle interactions is often problem-specific. In this section, we follow [11] for the description of the different contributions.

Mainly, the influence of fluid on particle motion can be described as the sum of two contributions: drag forces and lift forces.

3.1.1. Drag forces

The drag forces that arise on the particle due to the fluid are:

- Undistrubed flow
- Steady state drag
- Virtual (or added) mass
- Basset term

Undisturbed Flow

This component describes the contribution of the pressure and the shear stress fields in the undisturbed flow, i.e. in the flow without considering the presence of the particle. This term contributes significantly in liquid-particles flows, but it is negligible in gas-particle flows. The model is described by:

$$F_{ud} = V_d\left(-\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ik}}{\partial x_k}\right), \tag{3.1}$$

where V_d is the volume of the particle.

Steady state drag

This terms models the drag force that acts on the particle in a velocity field where there is no acceleration of relative velocity between the particle and the conveying fluid. Different models are available in the literature, to describe a variety of situations. We give, as an example, the steady state drag force based on the drag coefficient C_D:

$$F_{ss} = \frac{1}{2}\rho_c A C_D |u - v|(u - v), \tag{3.2}$$

where ρ_c is the density of the fluid carrier, A is the project area of the particle in the direction of relative velocity of fluid and particle and C_D is the drag coefficient. The drag coefficient C_D is strongly influenced by the Reynolds number Re. The dependence of C_D from Re has been extensively studied and there are several models available for different ranges of Re.

Virtual (or apparent) mass effect

The virtual mass effect appears when a body in a fluid is subjected to an acceleration. The fluid is accelerated, and the work necessary to obtain this acceleration is done by the body. Therefore, the global effect is called apparent mass force, since it is equivalent to adding a mass to the sphere.
The virtual mass force can be modeled as:

$$F_{vm} = \frac{1}{2}\rho_c V_d\left(\frac{Du}{Dt} - \frac{dv}{dt}\right), \tag{3.3}$$

being V_d the volume of the particle.
Virtual mass effect appears only in unsteady flows.

Basset Term

This term describes the delay in the development of boundary layers due to the fact that relative velocity changes with time. It describes the viscous effects of an acceleration and it is known in literature also as history term, since its value depends on acceleration history up to the present time. This contribution is modeled as:

$$F_{Bass} = \frac{3}{2}D^2\sqrt{\pi\rho_c\mu_c}\left[\int_0^t \frac{\frac{d}{dt'}(u_i - v_i)}{\sqrt{t - t'}}\,dt' + \frac{(u_i - v_i)_0}{\sqrt{t}}\right]. \tag{3.4}$$

We see that also the initial value of the relative velocity influences the final force F_{Bass}.
Basset term appears only in unsteady flows.

3.1.2. Lift forces

The lift forces that arise are:

- Magnus lift
- Saffman lift

Saffman Lift

The Saffman lift arises from the pressure distribution on a particle in a velocity gradient. Let us consider a sphere immersed in a shear flow, with higher velocity at the top of the particle and lower at the bottom. The higher velocity on at the top causes a low pressure, whereas the lower velocity at the bottom gives rise to a high pressure. This differential of pressure develops a lift force.
Let us define the shear Reynolds number Re_G, i.e the Reynolds number based on the velocity difference between the top and the bottom of the particle:

$$Re_G = \frac{D^2}{\nu_c}\frac{du}{dy}, \tag{3.5}$$

where ν_c is the kinematic viscosity of the fluid carrier and D is the diameter of the particle. If both the relative Reynolds number based on velocity different of fluid and particle Re_r and the shear Reynolds number are small ($\ll 1$) and $Re_r \ll \sqrt{Re_G}$, then the Saffman lift force can be modeled as:

$$F_{Saff} = 1.61\mu_c D|u_i - v_i|\sqrt{Re_G}. \tag{3.6}$$

Magnus Lift

The Magnus force is the lift generated by the rotation of a particle, due to the presence of a pressure differential between both sides of the particle, caused by the velocity differential due to rotation. The rotation may be caused by other phenomena than velocity gradient.
Generally, the direction of the force is the normal to the plane formed by the rotation vector and the relative velocity vector. If they are orthogonal to each other, the Magnus lift force can be modeled as:

$$F_{Mag} = \frac{1}{2}\rho_c A C_{LR}|v - u|(v - u), \tag{3.7}$$

where A is the project area of the particle and C_{LR} is the lift coefficient due to rotation. The modeling of the coefficient C_{LR} has been explored extensively and a variety of equations are available in literature.

3.1.3. Total interaction

Exploiting the same procedure adopted by Discrete Element Method in the modeling of particle/particle interactions, the total force is obtained summing all the contributions relevant for the specific problem. The total force exerted by the fluid on the particle is therefore:

$$F_{fp} = F_{ud} + F_{ss} + F_{vm} + F_{Bass} + F_{Saff} + F_{Mag}. \tag{3.8}$$

Applying third Newton's law we obtain that the force exerted by the particle on the fluid is equal in magnitude and opposite in direction to the total force that the fluid exerts on the particle that we just modeled.

3.2. General issues of Coupling

The coupling between CFD and DEM presents two intrinsic difficulties: an accurate choice for the time step of the simulation and an efficient approach for contact detections.

In fact, for the choice of time steps, we have to consider that it is necessary to catch collision dynamics and satisfy maximum particle overlap constraint (since particles deformation are not modeled directly, but through their overlap).

As far as contact detection is concerned, it is straightforward to notice that a detection of contacts at every time step for every particle is extremely expensive from a computational point of view and, of course, not optimal. The problem is how to detect which particles collide every time step in a computationally efficient way. Various approaches have been proposed and we will discuss the Neighbor List method developed by Verlet in 1967 and the link-cell method proposed by Plimpton in 1995.

3.2.1. Time step choice

It is really important to choose carefully the time steps for the phases of CFD and DEM. Usually, the time step for DEM is smaller than the one for CFD: in most cases the DEM-time step has to be at least an order smaller. In order to compare the two time scales, three important parameters are evaluated:

- CFL number for CFD,
- Rayleigh time for DEM,
- Particle relaxation time for CFD-DEM.

CFD

We consider only the presence of a fluid phase. For the numerical integration of the Navier-Stokes equations, the most important parameter to consider is the CFL number (Courant-Friedrichs-Lewy). In an n dimensional case, CFL is defined as:

$$CFL = \Delta t \sum_{i=1}^{n} \frac{u_i}{\Delta x_i}, \tag{3.9}$$

where Δt is the time step and Δx is the space step in the computational grid.
Since the CFL number is a measure of how many cells an infinitesimal volume of fluid passes in one time step, this has to be smaller than one in order to preserve the stability of the numerical scheme. Therefore, we get a constraint on the CFL which translates in a constraint on the time step Δt:

$$CFL = \Delta t \sum_{i=1}^{n} \frac{u_i}{\Delta x_i} < 1 \;\Rightarrow\; \Delta t < 1/\sum_{i=1}^{n} \frac{u_i}{\Delta x_i}. \tag{3.10}$$

DEM

Considering only the interactions between particles, we are in the field of granular flows.
In high density particle regions, the motion of particles is affected not only by forces and torques arising from collisions with particles in the immediate neighbor, but also by disturbances propagating from more distant particles. The propagation of these disturbances is modeled via Rayleigh waves, i.e. surface waves that travel (with both longitudinal and transverse components) near the surface of solids. To ensure realistic force transmission rates and to prevent numerical instabilities, an upper bound for the simulation time step is therefore necessary.

The idea proposed in [34] and [1] is that time step for detecting collision between a particle and its neighborhood should be less than the time it takes for the Rayleigh wave to transverse the minimum size particle in the assembly. The Rayleigh time step proposed is therefore:

$$T_R = \pi r \frac{\sqrt{\rho/G}}{0.1631\nu + 0.8766}, \tag{3.11}$$

where r and ρ are the radius and the density of the particle, G is the particle shear modulus and ν is Poisson's ratio. We see that the time-step is material dependent through G. Hence, if we want to model the motion of particles of different materials, we necessarily have to consider the minimum of the different Rayleigh time steps.
Therefore, to prevent numerical instabilities and nonphysical results, $\Delta t_{DEM} < T_R$. Often, a fraction of T_R is used for the integration time-step.

CFD-DEM: Particle Relaxation Time

Exploring the interaction between fluid phase and solid particles, the concept of particle relaxation time τ is introduced as a measure of the resistance of a particle to adapt to flow motion: the larger τ, the stronger the resistance. The definition of the particle relaxation time is not unique: depending on the specific problem, different models can be used.
In [11] particle relaxation time τ is modeled, in the Stokes regime ($Re \ll 1$), as:

$$\tau = \frac{\rho_p d^2}{18\mu}, \tag{3.12}$$

whereas in [18] another model is used:

$$\tau = \frac{\rho_p d^2}{18\mu}(1 + 0.15Re^{0.687})^{-1}, \tag{3.13}$$

In order to achieve stability of the numerical method, the DEM time step width has to be lower than particle relaxation time:

$$\Delta t < \tau \;\Rightarrow\; \Delta t < \frac{\rho_p d^2}{18\mu}(1 + 0.15Re^{0.687})^{-1}. \tag{3.14}$$

As we see, we have three constraints for two parameters (Δt for CFD and DEM). As a rule of thumb DEM-Δt usually at least one order smaller than CFD-Δt, but mainly the approach is problem-dependent and literature material does not provide satisfactory results.

3.2.2. Contact Detection Algorithm

In order to model accurately the dynamics of contacts, it is important to verify if two particles are colliding, at each time step of the simulation. As stated in the introductory paragraph, it is absolutely not efficient to check at every time step if each particle is colliding with every other particle in the system. This operation would cost n^2 checks, being n the number of particles. Since the check should be done at every time step of the simulation, the process would become too computationally expensive.

We now discuss an approach to overcome to this limitation: the Neighbor List method, proposed by Verlet. The main idea is the periodic construction of a list of potential contacts, in order to exclude *a priori* evaluations of contacts between particles too distant. Every time step, the algorithm checks the list for each particle and evaluates if that particle is colliding with the particles in its neighbor list. Of course, the list is built with a period larger than a simulation time step: we define N as the number of time steps after which the list is updated. In the hypothesis of spherical particles, we include a pair of particles (i and j) if it holds:

$$\|\mathbf{x}_i - \mathbf{x}_j\| \leq r_i + r_j + s, \tag{3.15}$$

where r_i and r_j are the radii of particles i and j respectively and s is the Verlet parameter, that can be chosen between some bounds. If we assume a constant time steps Δt and a maximum particle velocity v_{max}, the number of time steps after which we update the list can be modeled as:

$$N = \frac{s}{2v_{max}\Delta t}. \tag{3.16}$$

Alternatively, another method to build the neighbor list is called link-cell method and it is based on a binning approach on a grid decomposition. In this case, the parameter to choose is the length scale of the binning.

Comparison between the two approaches is explored in [31], but results are often problem dependent, since lots of parameters require careful tuning. The general trend is that in case of a relatively small number of particles is relatively small, the Verlet detection algorithm is faster than linked-cell algorithm, and the linked-cell algorithm seems more efficient when the number of particles is large.

3.3. Resolved CFD-DEM

In the resolved coupling we deal with particles which cover multiple cells of the CFD computational grid. The core idea of this method is to add a force term to the Navier-Stokes equations, in order to take under consideration the presence of the solid particles.

Since the particles are larger than the CFD computational grid, it is not possible to consider the presence of the particle in only one CFD cell (e.g. the cell where the centroid of the particle lies): this would lead to nonphysical results, in which the fluid would flow even in cells occupied entirely by the solid particle. To overcome this problem, ad hoc methods were developed in order to be able to resolve the fluid in an accurate way. These methods are known in literature as Immersed Boundary Methods (IB) and Fictious Domain Methods (FD).

Fictious domain methods are general techniques developed for solving differential equations on complex domains. The problem is translated into a simpler domain, solved and then the solution is corrected to satisfy the original problem. Instead, an Immersed Boundary Method is a specific approach for CFD to simulate fluid-structure interactions. Basically, Immersed Boundary Method belongs to Fictious Domain approaches. We will give two applications of these approaches to Resolved coupling, a fictious domain method presented in [18] and the PISO Immersed Boundary scheme developed in [6].

A Fictious domain Method

In this application of the Fictious domain approach the aim is to perform a correction of the velocity field of the fluid. This can be proved to be equivalent to adding a force term to the Navier-Stokes equations. The approach that we will describe provides satisfactory results for moderate Reynolds number. We consider only one velocity field and one pressure field in the domain and they are shared by the fluid and the solid. The domain taken into account is provided in Figure 3.1.

Firstly, only the fluid is considered and in the whole domain the velocity field is calculated from the following equations:

$$\begin{cases} \rho\left(\dfrac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) &= -\nabla p + \mu\Delta\mathbf{u} + \mathbf{f} & \text{in } \Omega_f \\ \nabla\cdot\mathbf{u} &= 0 & \text{in } \Omega_f \\ \mathbf{u} &= \mathbf{u}_\Gamma & \text{on } \Gamma \\ \mathbf{u}(x, t=0) &= \mathbf{u}_0(x) & \text{in } \Omega_f \\ \mathbf{u} &= \mathbf{u}_i & \text{on } \Gamma_s \\ \boldsymbol{\sigma}\cdot\hat{\mathbf{n}} &= \mathbf{t}_\Gamma & \text{on } \Gamma_s. \end{cases} \tag{3.17}$$

The first two equations are the standard Navier-Stokes equations for incompressible fluids. The third and the fourth ones are the boundary conditions on the entire domain and the initial conditions, respectively. The last two equations concern the actual coupling between the fluid and the solid phases: they provide the continuity of velocity field and the normal component of the stress tensor.

Once the data from DEM have been evaluated through a numerical integration, the method consists in the following four phases:

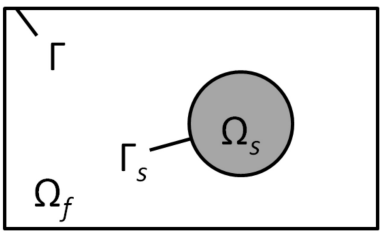


Figure 3.1: **Domain for Resolved Coupling**. The image is taken from [18], page 11

1. Evaluation of an interim velocity field $\hat{\mathbf{u}}$ and its associated pressure field
   The interim velocity is obtained solving the Navier-Stokes equation in the whole domain, usually using a finite volume method with the PISO (Pressure-Implicit with Splitting of Operators) algorithm.

2. Creation of a new velocity field $\tilde{\mathbf{u}}$
   The interim velocity is corrected in the particle areas imposing the velocity obtained from the DEM calculations.
   This step is equivalent to adding a force term $\mathbf{f}$ to the Navier-Stokes equation, where $\mathbf{f}$ satisfies:
   $$\mathbf{f} = \rho \frac{\partial}{\partial t}(\tilde{\mathbf{u}} - \hat{\mathbf{u}}). \tag{3.18}$$

3. Correction of the new velocity field into the final one $\mathbf{u}$
   Unfortunalety, the new velocity field does not satisfy the divergence-free constraint of the Navier-Stokes equation, which comes from mass conservation under the hypothesis of incompressibility. Therefore, the application of a correction operator is necessary.
   We define a corrected field $\mathbf{u}$
   $$\mathbf{u} = \tilde{\mathbf{u}} - \nabla\phi, \tag{3.19}$$
   where ϕ is an unknown scalar field and $\mathbf{u}$ is forced to satisfy the divergence-free constraint.
   Hence, applying the divergence operator to (3.19) we get a Poisson equation for ϕ:
   $$\Delta\phi = \nabla \cdot \tilde{\mathbf{u}}. \tag{3.20}$$

4. Correction of the pressure field
   After having solved (3.20), the pressure obtained from the first step can be corrected in the final one.

Hence, the global outline of the Fictious Domain Resolved method is:

Algorithm 1 Outline of Fictious Domain Resolved CFD-DEM coupling

```
procedure RESOLVED
    while (not done) do
        DEM solver: evaluation of positions and velocities of particles
        Data from DEM solver are passed to CFD solver
        Evaluation of interim velocity field
        Particle tracking: locate cells occupied by each particle
        Correction of velocity in the cells occupied by particles
        Evaluation of fluid forces acting on particles
        Data from CFD solver are passed to DEM solver for the next time step
        Divergence-free correction of velocity field
        Evaluation of other equations (i.e. concentrations, ...)
```

Immersed Boundary Method

The fictious domain approach presented in the last paragraph gives accurate results at moderate Reynolds numbers. According to [17], at low Reynolds numbers the results are not satisfactory anymore, so in 2015 another approach was proposed. This method is known as PISO Immersed Boundary and it was presented in [6]. It is a slightly more complicated method, since it is based not only on the correction of velocity and pressure fields, but also of the force term. The results are valid also in the case of low Reynolds numbers.

The idea of the PISO-IB scheme is to add an immersed boundary method to the standard PISO scheme. This is pursued using the PISO loops to impose the velocity of the immersed rigid body while maintaining mass conservation. In every iteration a continuous forcing term is updated and added to Navier-Stokes equation to take into account the immersed body and its motion. The method can be implemented and applied in parallel and it can be used with unstructured polyhedral meshes.

The flow of the algorithm is very similar the the approach developed using the Fictious domain Method. The main improvement is a correction of the forcing term which is performed at the end of every the PISO loop. The main structure of the algorithm is:

1. Detection of immersed body through cell and vertex flagging
   A solid fraction β_i is generated for every cell i
2. Evaluation of forcing term, based on previous data
3. Evaluation of interim velocity from a momentum predictor
4. Start of PISO loop
   - Correction of velocity
   - Solution of pressure correction equation
   - New correction of velocity
   - Correction of pressure
   - Correction of forcing term
     This is the main feature of the method. Forcing term is corrected using the difference between current velocity and the the velocity prescribed within the immersed body.

$$f_i^{new} = f_i^{old} + \frac{\alpha\beta}{\Delta t}(u_{i,ib} - u_{i,current}), \tag{3.21}$$

     where β_i is the solid fraction evaluated in step 1. and $\alpha \in]0, 0.9]$ is a relaxation parameter

General issues of Resolved Coupling

Resolved methods belong to the class of Direct Numerical Simulations (DNS), therefore, in order to have very precise results, a high resolution of the fluid mesh in the area of the particles is required. This constraint leads to enormous computational costs, even for small problems. Some remedies can be applied to overcome this limitation.

The first improvement is given by *dynamic local mesh refinement*. This process consists in a mesh refinement around the particles. When a particle moves on, the cells are coarsened again. Especially for dilute particle systems this has a large effect. This feature is already provided by OpenFOAM itself.

Another improvement can be *parallelization*, but in this case, particular attention is to be given in the communication between processors. A void fraction distribution model has been developed to take care of this issue.

3.4. Unresolved CFD-DEM

Unresolved CFD-DEM coupling deals with particles with sizes smaller than the CFD computational grid. Since different numbers of particles (of different sizes) can occupy one CFD grid cell, it is useful to introduce a new variable α, which represents the volume fraction occupied by the fluid in a cell. Therefore, we use the so-called locally averaged Navier-Stokes equations for the unknown $\mathbf{u}$, velocity of the fluid phase:

$$\begin{cases} \rho\left(\dfrac{\partial(\alpha\mathbf{u})}{\partial t} + \nabla\cdot(\alpha\mathbf{u}\mathbf{u})\right) & = -\alpha\nabla p + \mu\Delta\mathbf{u} + R_{pf} \\ \dfrac{\partial\alpha}{\partial t} + \nabla\cdot(\alpha\mathbf{u}) & = 0. \end{cases} \tag{3.22}$$

where R_{pf} is the force exchange term, i.e. it takes into account the interaction between the fluid and particle phases. It is evaluated as:

$$R_{pf} = K_{pf}(\mathbf{u} - \mathbf{u}_p), \tag{3.23}$$

where $\mathbf{u}_p$ is the velocity of the particle (taken from DEM data) and K_{pf} is a coefficient. The three major contributions on the interaction between fluid and particles are the gradient of pressure, viscous and drag force. Since the pressure gradient and the viscous term are already taken under consideration in the stress tensor, we model the coefficient K_{pf} using only the contribution of drag forces. Being V the volume of the cell, we define $f_{d,i}$ as the drag force acting on the fluid due to particle i and we get:

$$K_{pf} = \frac{\sum_i f_{d,i}}{V}. \tag{3.24}$$

Other forces may be relevant depending on the specific problem and they can be simply added in the expression of K_{pf}. For example, Magnus force for the rotation of particles, virtual mass force for particle acceleration, Saffman force for gradient of fluid velocity leading to shear. The modular feature of the coupling allows to consider these contributions, properly modeled, directly in the exchange term R_{pf} through K_{pf}.

As stated before, usually drag forces are the most relevant contributions and various techniques can be applied in the modeling of K_{pf}. One possibility has been presented in section 5.1, but other models have been developed. In fact, we now consider a combination of Wen and Yu model (for $\alpha > 0.8$) and Ergun model (for $\alpha \leq 0.8$), which we present in the following paragraph, but it is important to notice that the choice of the model is not unique, since several models are available. We define d as the diameter of the particle under consideration, the Reynolds number base on relative velocity Re_p and the drag coefficient C_d as:

$$Re_p = \frac{|\mathbf{u} - \mathbf{u}_p|\rho d}{\mu}, \tag{3.25}$$

$$C_d = \frac{24}{\alpha Re_p}[1 + 0.15(\alpha Re_p)^{0.687}]. \tag{3.26}$$

Hence, K_{pf} is modeled as:

$$K_{pf} = \begin{cases} C_d \frac{3}{4} \frac{\alpha(1-\alpha)|\mathbf{u} - \mathbf{u}_p|}{d} \alpha^{-2.65} & \alpha > 0.8 \\ 150 \frac{(1-\alpha)^2 \mu}{\alpha d^2 \rho} + 1.75 \frac{(1-\alpha)|\mathbf{u} - \mathbf{u}_p|}{d} & \alpha \leq 0.8 \end{cases} \tag{3.27}$$

The DEM equations are exactly those presented in the section concerning the DEM approach and they have to be solved before the CFD phase, in order to add to the locally averaged Navier Stokes equations the proper terms for the specific problem.

The outline of the complete algorithm is given by the following pseudo-code:

Algorithm 2 Outline of Unresolved CFD-DEM coupling

```
1:  procedure RESOLVED(a,b)
2:      while (not done) do
3:          DEM solver: evaluation of positions and velocities of particles)
4:          Data from DEM solver are passed to CFD solver
5:          Particle tracking: for each particle determine the cell in which it lies
6:          Determine particle volume fraction and mean particle velocity for each CFD cell
7:          Evaluation of fluid forces from particle volume fraction, for each particle
8:          Evaluation of exchange terms, for each cell
9:          Evaluation of fluid velocity (considering particle volume fraction and exchange terms), for each cell
10:         Data from CFD solver are passed to DEM solver for the next time step
11:         Evaluation of other equations (i.e. concentrations, ...)
```

Coarse Averaging procedures

One of the most challenging steps in this approach is the interpolation of a Lagrangian property, the particles volumes fraction, from the DEM side to an Eulerian property, the volume fraction field, which is defined on the fixed Eulerian grid of the CFD simulation. The same process has to be performed also for particle velocity and fluid-particle interaction force.

Hence we have to consider methods to interpolate the following physical quantities:

1. Solid volume fraction α,
2. Solid phase velocity $\mathbf{u}_p$,
3. Fluid–particle interaction force R_{pf}.

In the literature, this process is often called "coarse graining" or "averaging". We therefore require an interpolation and different approaches have been proposed. A sum up of the different methods is provided in [42] and [41]
The features that an ideal coarse graining procedure should have are:

1. Conserve relevant physical quantities
2. Handle particles both in the interior cells and the cells near boundaries without producing artifacts
3. Achieve relatively mesh-independent results
4. Be convenient for implementation in parallel
5. Produce smooth coarse grained fields even with the presence of a few large particles in relatively small cells

Some approaches developed for this purpose are Particle centroid method (PCM), Divided particle volume method (DPVM), Statistical Kernel method and the Diffusion-based coarse graining. For the sake of simplicity we will describe these methods for the scalar quantity α (solid particle fraction), but the approches can be generalized for the vector fields $\mathbf{u}_p$ or R_{pf}, component-wisely.

- Particle Centroid Method (PCM).
  It consists in summing over all particle volumes in each cell where the particle centroid lies, to obtain cell-based solid volume fraction. This method is easy to implement in CFD solvers, but it can lead to large errors when cell size to particle diameter ratios are small. Unphysical results can be obtained (for example, particle volume fraction greater than 1). In particular, we emphasize that the volume fraction is characterized by steep gradients, since the entire volume of a particle is added to the cell where the centroid lies, even if the centroid is near the border of the cell and therefore the actual volume would occupy different cells.

- Divided Particle Volume Method (DPVM),
  The volume of a particle is divided among all cells that it overlaps with, according to the portion of the volume within each cell. Hence, the solid volume fraction in any cell never exceeds 1 and large gradients in the obtained field are prevented. DPVM works for arbitrary meshes, as long as the particle diameter is smaller than CFD cell size. Also with this method, it is likely to obtain a steep gradient in the volume fraction field.

- Two grid formulation
  The idea is to use two independent meshes for the averaging and for the CFD simulation. The averaging mesh is chosen based on particle diameters to ensure that the cell sizes are larger than particle diameters. The CFD mesh is chosen according to flow resolution requirements.

- Statistical Kernel method
  The volume of each particle is distributed to the entire domain according to a weight function called kernel function $h(\mathbf{x})$. The solid volume fraction at location $\mathbf{x}$ consists of the superposition of the distributed volumes from all particles

- Diffusion-based coarse graining
  Firstly, an initial value $\alpha_0(\mathbf{x})$ is obtained using PCM. Then, a transient diffusion equation (in this case, the heat equation) for $\alpha(\mathbf{x}, t)$ is solved with initial condition $\alpha_0(\mathbf{x})$ and no-flux boundary conditions (i.e. homogeneous Neumann conditions), to ensure mass conservation. The following Cauchy problem is to solve:

$$\begin{cases} \dfrac{\partial \alpha_s}{\partial \tau} - \Delta \alpha_s & = 0 \\ \alpha_s(\mathbf{x}, 0) & = \alpha_s^0(\mathbf{x}) \quad \leftarrow \text{from PCM} \end{cases} \tag{3.28}$$

  The result, is a field $\alpha(\mathbf{x}, t)$ and it is the solid volume fraction field to be used in the CFD-DEM formulation. A critical parameter is the final time T for the solution of the diffusive equation, which has to be a physical parameter characterizing the length scale of the coarse graining. Diffusion equation is solved on the same mesh as the CFD mesh.

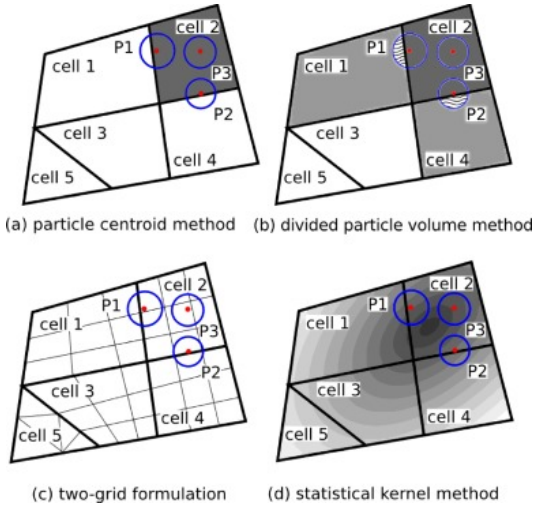


Figure 3.2: **Coarse graining methods**. The image, from [42], clarifies the differences in the coarse-graining methods described in the previous paragraph.

In Figure 3.2, a representation of the main features of the standard techniques for the coarse graining procedure is given. A detailed comparison between these graining procedure is available in the papers cited.

It may be important to highlight a connection between the diffusion-based method and the Statistical Kernel Method, if we base the latter on a Gaussian distribution. We remind that the diffusion-based method consists in considering the outcome of the Parcel Centroid method as initial condition, which is characterized by the presence of steep gradients in the field. The diffusion-based procedure is able to stabilize the result, since the diffusion equation essentially redistributes particle volumes within the field conserving automatically total solid volume in the domain through the diffusion.

In order to solve the diffusion equation numerically exploiting the already-built CFD mesh, we need to estimate a suitable time step τ. To this purpose we build an the equivalence between the diffusion-based coarse graining and a Statistical Kernel method based on the Gaussian distribution, exploiting the fact that the general solution of the heat equation (3.28) taken under consideration, is the convolution of a Gaussian distribution. It is possible to prove that the diffusion-based procedure is equivalent to a Gaussian Kernel method if the bandwidth of the Gaussian distribution b satisfies:

$$b = \sqrt{4\tau}, \tag{3.29}$$

where τ is the time step of the diffusion procedure. Imposing a value for the Gaussian bandwidth b, (typically in the literature $b = 6d$, being d the diameter of the particle), we obtain a value for the diffusion time step τ. Solving numerically the diffusive equation (3.28) for just 1 to 3 time steps τ, we obtain satisfactory results for the particle volume fraction field $\alpha_s(\mathbf{x})$.

Mesh convergence results and in-depth comparison of the performance of these approaches are explored in [42]. The diffusion method proposed by the authors manages to achieve very good results in both mesh-convergence and simplicity of implementation. Hence, an implementation of the coupling between CFD and DEM should be based on the diffusion coarse graining approach.

3.5. Resolved/Unresolved Coupling

Adopting one of the two techniques presented in the previous sections is not enough to simulate every physical process. It is of course possible to have situations in which both large and small solid particles are present and important. Therefore, one of the current research areas is to develop methods and algorithms in which both the cases are considered and solved. A first attempt to develop a strategy for the solution of the problem has been published in [24], in which an algorithm has been developed for an implementation on ANSYS/Fluent platforms.

The coupling developed is a combination of the resolved and unresolved approaches. The particle-fluid interaction are considered at different scales: cell level for fluid phase and particle level for particulate phase.

The idea is, at each CFD time step, to evaluate firstly the fluid forces acting on small particles using a DEM solver to take into account all the inter-particle forces, the influence of walls and the influence of large objects, using data from the previous CFD time step. These evaluations are done until the time of CFD simulation is reached.
After we have reached the synchronization between CFD time and small-particles DEM time, we can move forward on the algorithm and start the evaluation of large-particles dynamics. The calculation of cell volume fraction and momentum source terms is performed through the evaluations of the influences of fluid flow, small particles and walls on the large particles.
This process is performed until the final time of simulation is reached.

In Figure 3.3 we report a numerical result obtained in [24], to give an idea of the capability of the approach. Particle distribution in a fluidized bed with an immersed free-moving tube at different times is plotted

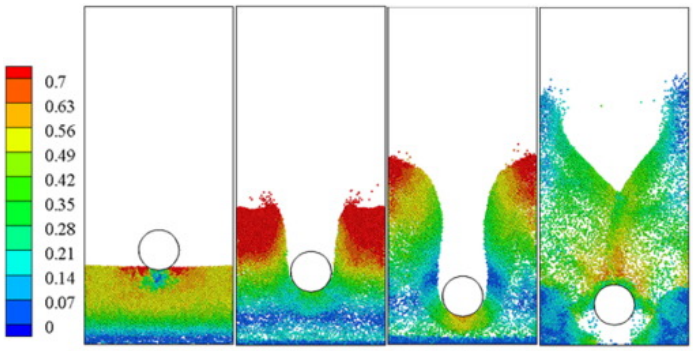


Figure 3.3: **Fluidized bed with an immersed free-moving tube** The image is taken from [24]. Particle distribution is plotted and particles are colored by velocity magnitude.

3.6. Current implementations

Some libraries are available to simulate the (resolved or unresolved) coupling between fluid and particle simulation.

- `DPMFoam` is a library provided by the standard distribution of OpenFOAM. It does not implement the Discrete Element Method, but DPM (Discrete Parcel Method). The idea is to aggregate clouds of particles and treat them as one big computational particle. This method has strong limitations: the dynamic properties (size, velocity, restitution coefficient, density, ...) for each particle in the parcel have to be the same and the dynamic is solved only at cloud level, not at particle level.

- `sediFoam` is a library developed by Sun and Xiao (2015), for CFD-DEM unresolved coupling in OpenFOAM. The implementation has been described in [43]. The main focus of this library is the simulation of sediment transport and fluidized beds. The most peculiar feature provided in this coupling is the diffusion-based coarse graining approach to perform the evaluation of particle volume fraction, as described in the previous section. The method seems stable for particles sizes at most two times the CFD cell size. The library couples OpenFOAM 2.4.0 with LAMMPS 1-Fed-2014.

- CFDEM is a set of libraries which probably provide the most complete implementation of the coupling. The software is provided by DCS Computing in open source and commerical version. It has been developed both for resolved and unresolved coupling. The solver which has been tested in the outlook of this work is `cfddemsolverIB`. In the resolved approach the Fictious Domain Method described in section 3.3 has been implemented. In the unresolved approach a standar technique has been considered. It couples OpenFOAM-5.x and LIGGGHTS-3.8.0

- `coupledPimpleFoam` is a library developed by Dynaflow Research Group as an expansion of the standard `PimpleFoam` library provided by OpenFOAM. It features a simple adding term to the Navier Stokes

equation to take into account forces arising from the presence of particles in the flow. This method can be considered as unresolved. Some severe limitations of this library are:

1. DEM update is performed in every CFD step: this causes constraints on the CFD time step, since DEM time step usually has to be very small
2. Particle fraction field α is not evaluated. Various drag models are based on this quantity, therefore they cannot be considered
3. The coarse graining procedure is intrinsically PCM, therefore huge gradients can be present in the forcing term in the momentum equation.

Research Questions

The previous Chapters had the purpose to study and describe the models and methods available for fluid/-particle simulations. The field of numerical simulations of fluid/particle system is relatively new and therefore improvements are developed continuously, mostly in the latest years. New methods are proposed and implemented every months, making this topic particularly thriving and inspiring. In particular, during the literature study, some interesting research questions rose that we now list and discuss:

1. *How to improve the numerical integration algorithm used in the particle solver?*
   The most common integration algorithm implemented is the Stormer-Verlet scheme. From the point of view of Numerical Analysis, it would be interesting to study the reasons for its widespread usage as numerical integrator and to propose alternative algorithms that could improve performance and accuracy of the simulations.

2. *How to use the not-fixed ΔT_{DEM} of HADES in the coupling?*
   In the implementation of CFDEM, the particle solver is LIGGGHTS, which is characterized by the constraint of a *a priori* fixed ΔT_{DEM}. HADES, instead, allows non-fixed time steps and this could lead to severe improvements in the execution time of the simulations.

3. *How to handle non-spherical particles?*
   As stated in the appropriate Chapter, DEM is a powerful method to model interactions between spherical particles. Real-life simulations involve more complex geometries and scenarios, hence a generalization of the method to handle non-spherical particles would improve dramatically the accuracy of the results.

In the next Part of the Thesis, we will focus on these research questions, going further with theoretical study of the problems as well as developing new features to improve the current implementations available. In particular, we will focus on the first research question with a theoretical study on numerical algorithms in Chapter 4. The second one will be dealt with in Chapters 5 and 6 describing in detail the coding that was necessary to the substitution of LIGGGHTS with HADES in the CFDEM framework. Finally, the third question will be studied in Chapter 7, both from a theoretical point of view and the implementation side.

II

Developments

4

Numerical Methods for Symplectic Hamiltonian Systems

In this Chapter, we study how to solve numerically the equations of motion that arise from particle dynamics. Firstly, we introduce some simple numerical approaches, then we introduce the Störmer-Verlet Method, which is the approach that is extensively adopted in the implementations available. We collocate it in the framework of the partitioned Runge Kutta Methods and we discuss its most important property, the *simplecticity*, if it is applied to Hamiltonian systems. To this purpose, a brief introduction to Hamiltonian system is provided. Then we focus on the property of simplecticity. Finally, convergence behavior of the Störmer-Verlet method is tested and compared to simple numerical methods and Runge Kutta approaches. Implementation of the numerical schemes is performed in MATLAB.

As reference literature for these topics, we followed mostly [20] and [19], which provide an extensive discussion on the various features of numerical methods for ODEs and [22] for details on Geometric Numerical Integration. As complementary framework, we used [38] for general aspects of Numerical Analysis, [21] for a precise overview of the Störmer-Verlet Method and [12] for Symplectic Integrators.

4.1. Numerical Methods

4.1.1. Classical Numerical Methods

We now give a brief discussion on the numerical methods that were developed for the solution of differential equations. We introduce the Euler Methods (Explicit and Implicit), the partitioned Euler and the Störmer-Verlet scheme.

Euler Methods

Let us consider the following system of differential equations:

$$\begin{aligned} \dot{y} &= f(t, y) \\ y(0) &= y_0, \end{aligned} \tag{4.1}$$

where $y \in \mathbb{R}^d$. We denote with y_i the solution $y(t)$ at time i. Then the Euler methods read as following:

Explicit Euler Method $\quad y_{n+1} = y_n + hf(y_n)\,.$

Implicit Euler Method $\quad y_{n+1} = y_n + hf(y_{n+1})\,.$

It is straightforward to notice that in the explicit Euler method, the evaluation of the function f is performed at time n, i.e. the function is known at that time. Whereas in the Implicit Euler method, the function is evaluated at time $n+1$, i.e. the value is not known. This translates in the necessity to solve a (non-)linear

system at each time step. nevertheless, the higher computational demand is compensated by better stability properties of the algorithm.

Partitioned Euler Method

We now consider the following partitioned system of differential equations:

$$\dot{u} = f(u, v) \tag{4.2}$$
$$\dot{v} = g(u, v). \tag{4.3}$$

In this case, a combination of the Euler methods is possible: we treat one variable in an implicit way and the other in a explicit way. This leads to the following partitioned Euler Method, named also *Symplectic Euler Method* for its property of symplecticity, which will be analyzed in detail in the next Section.

$$u_{n+1} = u_n + hf(u_{n+1}, v_n) \tag{4.4}$$
$$v_{n+1} = v_n + hg(u_{n+1}, v_n). \tag{4.5}$$

Störmer-Verlet Scheme

Finally, we consider the second order ordinary differential equation:

$$\ddot{q} = f(q), \tag{4.6}$$

which arises very frequently in the modeling of physical system, e.g. Newton's Law. With a change of variables, a system of first order differential equations is easily obtained.

$$\dot{q} = p \tag{4.7}$$
$$\dot{p} = f(q). \tag{4.8}$$

A very important numerical scheme was developed to solve this systems and it is known is literature with different names, depending on the application field: in molecular dynamics is called *Verlet Method*, in numerical methods for PDEs *leap-frog Method*, in astronomy *Störmer Method*. We will refer to it as *Störmer-Verlet Method*.

Different formulations are available for the actual computations. The two steps formulation consists in the approximation of the second derivative with a central scheme. Geometrically speaking, this formulation interpolates the solution in a parabola which has the correct value of the second derivative at the mid point of the interval.

Two steps formulation:

$$q_{n+1} - 2q_n + q_{n-1} = h^2 f(q_n). \tag{4.9}$$

It is possible to build a more robust and stable one step formulation with the partial update of the value of one variable and the successive evaluation of the other variable considering the just-evaluated value.

One step formulation:

(I)
$$\begin{aligned} p_{n+1/2} &= p_n + \frac{h}{2} f(q_n) \\ q_{n+1} &= q_n + h p_{n+1/2} \\ p_{n+1} &= p_{n+1/2} + \frac{h}{2} f(q_{n+1}). \end{aligned} \tag{4.10}$$ (II)

$$\begin{aligned} q_n &= q_{n-1/2} + \frac{h}{2} p_{n-1/2} \\ p_{n+1/2} &= p_{n-1/2} + h f(q_n) \\ q_{n+1/2} &= q_n + \frac{h}{2} p_{n+1/2}. \end{aligned} \tag{4.11}$$

In both cases, combining the last step of the scheme with the first of the successive one, we obtain the scheme:

$$\begin{aligned} p_{n+1/2} &= p_{n-1/2} + hf(q_n) \\ q_{n+1} &= q_n + hp_{n+1/2}, \end{aligned} \tag{4.12}$$

which is stable, effective and computes the values of the solution in the *time grid* and the values of its first derivative in the mid-points of the grid.

4.1.2. Runge-Kutta Methods

Runge-Kutta methods are a family of numerical schemes developed for the solution of non-autonomous systems of first-order differential equations. They belong to the *one step methods* set, i.e. the solution at time t_{n+1} will depend only on time t_n and not on previous ones. The idea is to subdivide the interval in $[t_n, t_{n+1}]$ in s subintervals, where s is the number of *stages*.

We consider the following first-order differential equation:

$$\begin{aligned} \dot{y} &= f(t, y) \\ y(0) &= y_0, \end{aligned} \tag{4.13}$$

A general Runge Kutta method is given by:

$$y_{n+1} = y_n + hF(t_n, y_n, h; f), \tag{4.14}$$

where h is the time step, F is the incrementing function and K_i are the coefficients of the stages:

$$F(t_n, y_n, h; f) = \sum_{i=1}^{s} b_i K_i \tag{4.15}$$

$$K_i = f(t_n + c_i h, y_n + h \sum_{j=1}^{s} a_{ij} K_j). \tag{4.16}$$

Since the set of coefficients a_{ij}, b_i and c_i uniquely determines the Runge Kutta Methods, their values are usually shown using the *Butcher tableaus*.

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

We remark that, from literature, coefficient are required to satisfy the relation:

$$c_i = \sum_{j=1}^{s} a_{ij} \quad \forall i = 1, ..., s \tag{4.17}$$

Depending on the values of the coefficients, Runge Kutta methods can be:

- *explicit* if $a_{ij} = 0$ for $j \geq i$, $i = 1, ..., s$
- *semi-implicit* if $a_{ij} = 0$ for $j > i$, $i = 1, ..., s$ (in literature, also *diagonally implicit, DIRK*)
- *implicit* otherwise

Explicit methods have the strong advantage to compute the coefficients K only exploiting the previously-evaluated coefficients. Implicit methods require to solve a non linear system of dimension s for every coefficient (s times), whereas the semi-implicit methods require to solve 1 non linear equation for every coefficient (s times).

It is straightforward to obtain that the Explicit and Implicit Euler belong to the class of Runge Kutta Methods with the following Butcher tableaus:

Explicit Euler

0	0
	1

Implicit Euler

1	1
	1

Consistency, stability and convergence properties of Runge Kutta methods have been studied in detail in the XX century and in Section 4.4 we state a brief summary of the results.

Unfortunately, symplectic Euler Methods and the Störmer-Verlet Method do not belong to the class of Runge Kutta methods. In the next Section, we consider an extension of the Runge Kutta methods, i.e. the *partitioned Runge Kutta Methods*.

4.1.3. Partitioned Runge Kutta methods

We consider the following partitioned system of first-order differential equations:

$$\begin{aligned} \dot{y} &= f(t,y,z) \\ \dot{z} &= g(t,y,z). \end{aligned} \tag{4.18}$$

We decide to apply different Runge Kutta schemes to the two equations: the triplet (a_i, b_i, c_i) describes the first method, the triplet $(\hat{a}_i, \hat{b}_i, \hat{c}_i)$ the second one. A general partitioned Runge Kutta method is given by:

$$y_{n+1} = y_n + hF(t_n, y_n, z_n, h; f) \tag{4.19}$$

$$z_{n+1} = z_n + hG(t_n, y_n, z_n, h; g), \tag{4.20}$$

where h is the time step, whereas the incrementing functions F and G and the coefficients K_i and L_i are given by:

$$F(t_n, y_n, z_n, h; f) = \sum_{i=1}^{s} b_i K_i \tag{4.21}$$

$$G(t_n, y_n, z_n, h; g) = \sum_{i=1}^{s} \hat{b}_i L_i \tag{4.22}$$

$$K_i = f(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij} K_j, z_n + h\sum_{j=1}^{s} \hat{a_{ij}} L_j) \tag{4.23}$$

$$L_i = g(t_n + \hat{c}_i h, y_n + h\sum_{j=1}^{s} a_{ij} K_j, z_n + h\sum_{j=1}^{s} \hat{a_{ij}} L_j). \tag{4.24}$$

Using the following Butcher tableaus we can obtain the symplectic Euler method and the Störmer-Verlet scheme that we introduced in Section 4.1.1.

The Symplectic Euler scheme is obtained by combining the Implicit Euler and the Explicit Euler:

1	1
	1

0	0
	1

The Störmer-Verlet scheme is obtained by using the following Butcher tableaus, which correspond to the implicit trapezoidal rule and a particular case of LobattoIIIB (which belongs to the class of adaptive Runge Kutta methods):

0	0	0
1	1/2	1/2
	1/2	1/2

1/2	1/2	0
1/2	1/2	0
	1/2	1/2

The Störmer Verlet Method was adopted extensively in science and engineering due to a very important property, the *simplecticity*. Simplecticity is a property of mappings that deals with area preservation in $\mathbb{R}^2$ and projected area preservation in $\mathbb{R}^{2d}$ if $d > 1$ and it has a strong link to Hamiltonian systems. In fact, it can be proved that a system is (locally) Hamiltonian if and only if its flow is symplectic for all points in an appropriate neighborhood. The proof of this theorem can be found in [22], Chapter VI.2.

Störmer Verlet Method allows the connection between the Simplectic Integrators for Hamiltonian systems that arise from models in several disciplines and the (Partitioned) Runge Kutta Methods, therefore its role has been relevant and it became a milestone in the numerical methods panorama.

In the following Sections, we firstly introduce the notion of Hamiltonian systems and then we discuss the property of simplecticity.

4.2. Symplecticity of Hamiltonian Systems

As stated in the previous section, symplecticity is deeply connected to Hamiltonian Systems. Therefore, an introduction to Hamiltonian Systems is required.

4.2.1. Hamiltonian Systems

The equations of motion were originally proposed by Newton with the purpose to describe the movement of free mass points. Generalizations of those equations were developed by Lagrange: the dynamic of rigid bodies and bodies connected by springs could be studied and analyzed. Some years later, Hamilton proposed a simplification of the Lagrange approach, introducing important symmetries in the formulation.

We now give an introduction to the Lagrangian and Hamiltonian Canonical forms.

The Lagrangian formulation requires *generalized coordinates* $q = (q_1, ..., q_d)$, where d is the degree of freedom of the system. Two important quantities need to be described:

- Kinetic energy $T = T(q, \dot{q})$
- Potential energy $U = U(q)$

With these two quantities, we define the Lagrangian L as:

$$L = T - U \tag{4.25}$$

which satisfies the following differential equation, named *Lagrange equation*:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) = \frac{\partial L}{\partial q}. \tag{4.26}$$

This approach provides a powerful framework to the modeling and solving of equations that arise from motions of bodies, but improvements were achieved by Hamilton, in 1834. In his work, he introduced two new quantities:

- Conjugate momenta $p = \dfrac{\partial L}{\partial \dot{q}}(q, \dot{q})$
- Hamiltonian $H = H(p, q) = p^T \dot{q} - L(q, \dot{q})$

With these new mathematical entities, it can be proved that the Lagrange equations (4.26) are equivalent to the following system, called *Hamiltonian Canonical equations*:

$$\begin{aligned} \dot{p} &= -\frac{\partial H}{\partial q}(p, q) \\ \dot{q} &= \frac{\partial H}{\partial p}(p, q). \end{aligned} \tag{4.27}$$

Hamiltonian Systems have two fundamental properties:

- the Hamiltonian $H = H(p, q)$ is a first integral of the system, i.e. it is *invariant*
- (projected) oriented areas are preserved (*simplecticity*)

4.2.2. Simplecticity

Simplecticity is a property that can be held by mappings. We firstly consider linear mappings and then we move to nonlinear ones. Finally, we will connect the concept of simplecticity to the family of Hamiltonian Systems, stating a fundamental Theorem of Geometric Numerical Integration.

To introduce simplecticity, we consider mappings that act on 2D parallelograms that lie in $\mathbb{R}^{2d}$ and that are spanned by the following two vectors:

$$\xi = \begin{bmatrix} \xi^p \\ \xi^q \end{bmatrix} \qquad \eta = \begin{bmatrix} \eta^p \\ \eta^q \end{bmatrix},$$

where $\xi^p, \xi^q, \eta^p, \eta^q \in \mathbb{R}^d$. The parallelogram lies in the (p,q) space, i.e. it is a linear combination of the vectors ξ and η.

If $d = 1$, we consider the oriented area of P, which is given by

$$\omega(\xi,\eta) := \det \begin{bmatrix} \xi^p & \eta^p \\ \xi^q & \eta^q \end{bmatrix} = \xi^p \eta^q - \xi^q \eta^p$$

Whereas if $d > 1$, we consider the sum of the projected oriented areas into the plane (p,q):

$$\omega(\xi,\eta) := \sum_{i=1}^{d} \det \begin{bmatrix} \xi_i^p & \eta_i^p \\ \xi_i^q & \eta_i^q \end{bmatrix} = \sum_{i=1}^{d} (\xi_i^p \eta_i^q - \xi_i^q \eta_i^p).$$

Defining the matrix $J \in \mathbb{R}^{2d}$ (let I be the identity matrix $\in \mathbb{R}^d$):

$$J = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix},$$

we can write the mapping $\omega(\xi,\eta)$ as:

$$\omega(\xi,\eta) = \xi^T J \eta. \tag{4.28}$$

We are now ready to define symplecticity for a linear mapping.

Let A be a linear mapping $A : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$. The mapping is called symplectic if

$$A^T J A = J \qquad \text{or} \qquad \omega(A\xi, A\eta) = \omega(\xi,\eta) \quad \forall \xi,\eta \in \mathbb{R}^{2d}. \tag{4.29}$$

With this definition, it is straightforward to notice that symplecticity corresponds to the conservation of areas in $\mathbb{R}^2 (d = 1)$, whereas it corresponds to the conservation of the projections of the oriented areas in higher dimensions into the (p,q) plane.

Symplecticity can be generalized to non-linear mappings. The idea is that each differentiable mapping can be approximated locally by linear mappings trough the Jacobian matrices.

Let O be an open set contained in $\mathbb{R}^{2d}$. Let g be a differential mapping $g : O \to \mathbb{R}^{2d}$. The mapping is called symplectic if the Jacobian matrix $g'(p,q)$ is everywhere symplectic, i.e. :

$$g'(p,q)^T J g'(p,q) = J \qquad \text{or} \qquad \omega(g'(p,q)\xi, g'(p,q)\eta) = \omega(\xi,\eta) \quad \forall \xi,\eta \in \mathbb{R}^{2d}. \tag{4.30}$$

This basic concepts have been extensively studied and despite their development is beyond the scope of this project, we give an important result which allows to understand why numerical algorithms which can preserve symplecticity of the problem have been researched and developed.

To this purpose, we need an elementary definition: the *flow* of a system. The flow of a system is a mapping that associates the value of the solution $y(t)$ to the solution with initial condition $y(0) = y_0$, i.e.

$$\varphi_t(y_0) = y(t) \qquad \text{if} \qquad y(0) = y_0. \tag{4.31}$$

Then, we are ready to state the theorem that links the concept of symplecticity to the concept of Hamiltonian systems:

Let O be an open set included in $\mathbb{R}^{2d}$. Let $f : O \to \mathbb{R}^{2d} \in \mathscr{C}^1$. Then $\dot{y} = f(y)$ is locally Hamiltonian if and only if the flow of the system φ_t is symplectic $\forall\, y \in O$ and for sufficiently small t.

This result is extremely important since it gives equivalence conditions for symplecticity of mappings and Hamiltonian systems. The next step is to develop numerical methods that can maintain the symplecticity of the problem, in order to obtain accurate results simulating the systems in their essence, preserving (projected) areas during the evolution in time.

4.3. Symplectic Numerical Methods

In [22], a basic and immediate definition of symplectic numerical methods is provided: a numerical method is called *symplectic* if the one-step map from y_n to y_{n+1} is symplectic, for all sufficiently small step sizes, when applied to a smooth Hamiltonian system.

We now give a list of conditions on the symplecticity of numerical methods according to the definition that we just stated:

- Partitioned Euler Method is symplectic
- Störmer-Verlet method is symplectic
- Runge-Kutta Methods: if the coefficients of the method satisfy:

$$b_i a_{ij} + b_j a_{ji} = b_i b_j \qquad \forall i, j = 1, ..., s, \tag{4.32}$$

  then the method is symplectic.

- Partitioned Runge-Kutta Methods: if the coefficients of the method satisfy:

$$b_i \hat{a}_{ij} + \hat{b}_j a_{ji} = b_i \hat{b}_j \qquad \forall i, j = 1, ..., s \tag{4.33}$$

$$b_i = \hat{b}_i \qquad \forall i = 1, ..., s, \tag{4.34}$$

  then the method is symplectic.

4.4. Convergence of Methods

In this Section we implement and test the numerical schemes presented in this Chapter. We choose two very simple problems to analyze the convergence behavior of the schemes. We test two different situations: a first order-differential equation and a partitioned system of differential equations. All the numerical schemes have been implemented in MATLAB.

The first step is to give an accurate definition of consistency, convergence and order of a numerical method.

A numerical method is said to be *consistent* if the exact solution of the problem satisfies the numerical method for the time step approaching to 0. We now write this definition in mathematical form. For the sake of simplicity, we consider a uniform grid, i.e. we consider $\tau = t_{n+1} - t_n$ as constant integration time step. Let $y(t_{n+1})$ be the exact solution evaluated in t_{n+1} and y^*_{n+1} be the numerical solution that corresponds to t_{n+1}, which is evaluated from the exact solution at time t_n. Then consistency (of order 1) is equivalent to the condition:

$$\frac{||y(t_{n+1}) - y^*_{n+1}||}{\tau} \leq C\tau. \tag{4.35}$$

In particular, denoting with $F(t_n, y_n, \tau; f)$ the incrementing function, consistency reads:

$$\frac{||y(t_{n+1}) - y(t_n) - \tau F(t_n, y_n, \tau; f)||}{\tau} \leq C\tau. \tag{4.36}$$

and C is a constant independent of the time step τ.
In particular, the method is said to be *consistent of order p* if

$$\frac{||y(t_n) - y^*_n||}{\tau} \leq C\tau^p. \tag{4.37}$$

Another fundamental concept is the convergence of numerical methods. A numerical method is said to be *convergent* if the distance between the exact solution and the numerical solution tends to 0 for the the time step approaching to 0. Mathematically, this is equivalent to:

$$||y(t_n) - y_n|| \leq C\tau, \tag{4.38}$$

where, again, the constant C is independent of the time step τ.
In particular, the method is said to be *convergent of order p* if

$$||y(t_n) - y_n|| \leq C\tau^p \quad \forall n. \tag{4.39}$$

Afterwards, we introduce the concept of *stability* of numerical methods: a numerical method is defined stable if small perturbations in the solution correspond to small perturbations in the data. An important result in Numerical Analysis states that if a numerical method is consistent, then it is convergent if and only if it is stable (a version of this Theorem is know as *Lax Equivalence*). For some numerical methods, stability can be proved assuming Lipschitz continuity, but this condition is too strict to be useful in general situations. If the Theorem is satisfied, the orders of convergence and consistency are maintained. In this case, we just refer to the method as a *method of order p* and we write $\mathcal{O}(\tau^p)$, considering the asymptotical behavior $\tau \to 0$.

It is straightforward to notice that, the higher the order of a numerical method, the better the accuracy of the simulation for the time step $\tau \to 0$.

Convergence analysis of numerical methods plays a fundamental role in Numerical Analysis and it is usually performed through Taylor expansions applied to the numerical schemes. A detailed discussion of the convergence behavior of numerical methods would require an extensive effort and goes beyond the scope of this project, therefore we simply list results available in literature for the schemes that we considered.

For the general first order differential equation systems:

- Explicit Euler is of order 1, $\mathcal{O}(\tau)$
- Implicit Euler is of order 1, $\mathcal{O}(\tau)$
- Implicit Trapezoidal is of order 2, $\mathcal{O}(\tau^2)$
- A general Runge Kutta Method is of order 1, $\mathcal{O}(\tau)$
  if $\sum_i b_i = 1$
- A general Runge Kutta Method is of order 2, $\mathcal{O}(\tau^2)$
  if $\sum_i b_i = 1$, $\sum_i b_i c_i = 1/2$
- A general Runge Kutta Method is of order 3, $\mathcal{O}(\tau^3)$
  if $\sum_i b_i = 1$, $\sum_i b_i c_i = 1/2$, $\sum_i b_i c_i^2 = 1/3$, $\sum_i b_i a_{ij} c_j^2 = 1/6$
- The Method Runge Kutta RK4, which will be introduced in the next paragraph, is of order 4, $\mathcal{O}(\tau^4)$.

For partitioned systems:

- Partitioned Euler is of order 1 $\mathcal{O}(\tau)$
- Störmer-Verlet is of order 2 $\mathcal{O}(\tau^2)$
- A general partioned Runge Kutta Method is of order 2, $\mathcal{O}(\tau^2)$
  if $\sum_{i,j} b_i \hat{a}_{ij} = 1/2$, $\sum_{i,j} \hat{b}_i a_{ij} = 1/2$
- A general partioned Runge Kutta Method is of order 3, $\mathcal{O}(\tau^3)$
  if $\sum_{i,j} b_i \hat{a}_{ij} = 1/2$, $\sum_{i,j} \hat{b}_i a_{ij} = 1/2$, $c_i = \hat{c}_i$, $\sum_{i,j} b_i \hat{a}_{ij} c_j = 1/6$, $\sum_{i,j} \hat{b}_i a_{ij} c_j = 1/6$

Further conditions and proofs for the results are available in [20] and [19].

These theoretical results are extremely important to test the implementation of numerical schemes. In fact, in the next paragraph we will verify the implementations of the methods through simple test problem for which an analytical solution is available, such that the evaluation of the numerical error is possible.

General Systems

Firstly, we focus on the first-order differential equation. We consider the general Initial Value Cauchy problem (IVP):

$$\begin{cases} \dot{y} = f(t, y) \\ y(0) = a. \end{cases} \tag{4.40}$$

In particular, we choose the following setting:

$$\begin{cases} \dot{y} = \cos(t) + y \quad t \in \langle 0, 1 \rangle \\ y(0) = 0. \end{cases} \tag{4.41}$$

The exact solution reads:

$$y(t) = \frac{e^t}{2} + \frac{\sin(t) - \cos(t)}{2}. \tag{4.42}$$

We compare the convergence behavior of the following methods:

- Explicit Euler
- Implicit trapezoidal (DIRK)
- RK4 (explicit)

The Implicit trapezoidal and RK4 are characterized by the following Butcher tables:

0	0	0
1	1/2	1/2
	1/2	1/2

0	0	0	0	0
1/3	1/3	0	0	0
2/3	−1/3	1	0	0
1	1	−1	1	0
	1/8	3/8	3/8	1/8

Firstly, we visualize in Figure 4.1 the exact and the numerical solutions obtained by applying the Explicit Euler scheme and the RK4 scheme. We adopt a coarse time step $\tau = 0.1$ and we observe that the two methods achieve different level of accuracy on the numerical solution. In particular, RK4 approximate the solution better than the Explicit Euler method. It is interesting to verify the convergence behavior of the methods to validate the implementation.

In order to study the convergence behavior of the numerical schemes, we adopt the *Richardson Method*, i.e. we start with a coarse time step τ and we sequentially divide it by 2. If the method is of order one, the error will be halved between the two simulations. If, instead, the method is of order 2, the error will be divided by 4. In general, if we divide the time step τ by n, the error will be divided by n^α, where α is the order of the method. Using this approach, we can study the order of convergence visualizing the behavior of the error and comparing it with τ to the power of n. For the system (4.41), using the Infinity norm for the measure of the error, we obtain Figure 4.2. Comparing the slopes, it is straightforward to notice that, as expected from the theoretical results, RK4 gives the best convergence behavior with order 4. The Implicit Trapezoidal method has order 2, whereas Explicit Euler has order 1. This is in accordance with the theory, therefore the implementation is validated. Another way to apply the *Richardson Method* to test the convergence is to evaluate the so called *Experimental Order of Convergence*, or EOC, between two consecutive errors obtained from consecutive τ. The EOC is evaluated as:

$$EOC_j = \frac{\log(err_j) - \log(err_{j+1})}{\log(\tau_j) - \log(\tau_{j+1})}. \tag{4.43}$$

The experimental order of convergence are listed, with the different τ used in the simulation. Note that since every evaluation of the order uses two different values of the error from two different τ, the vectors have one entry less than the vector of τ.

Tau	eoc_ExE	eoc_ImT	eoc_RK4
0.1	0.94575	2.0026	3.9616
0.05	0.97187	2.0007	3.9808
0.025	0.98567	2.0004	3.9904
0.0125	0.99277	2.0007	3.9952
0.00625	0.99637	1.9968	3.9968
0.003125	0.99818	1.9826	4.0322
0.0015625			

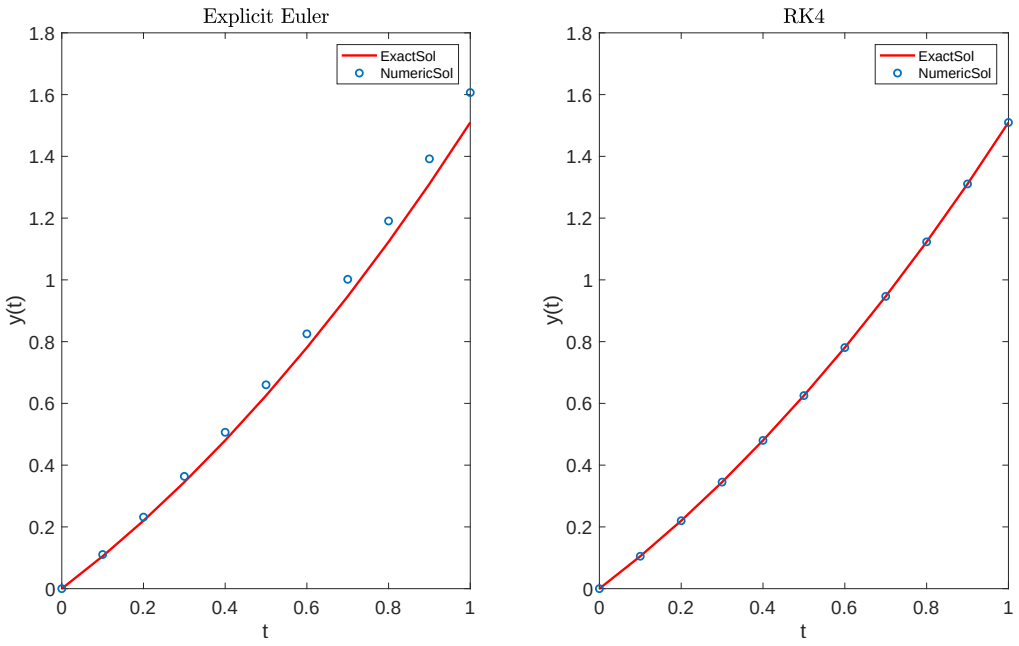


Figure 4.1: **Comparison between Explicit Euler and RK4** $\tau = 0.1$. The red straight line represents the exact solution, the dotted blue line the numerical solution. Accuracy is compared between the Explicit Euler Method and RK4. With a coarse $\tau = 0.1$, RK4 manages to approximate the solution better than Explicit Euler.

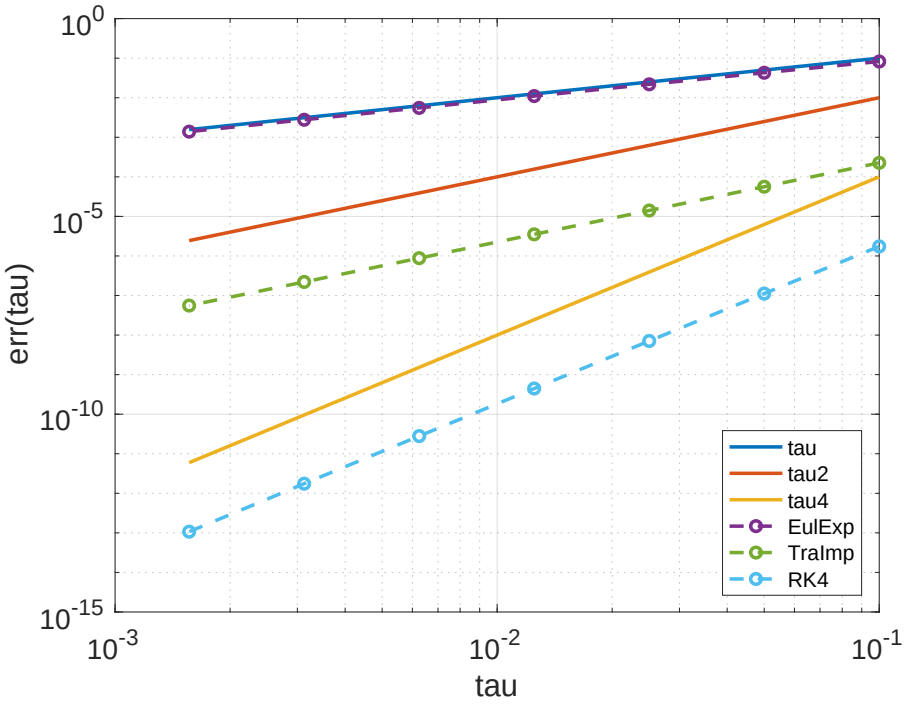


Figure 4.2: **Convergence of Numerical Methods**. The straight lines represent the behaviour of the time step to the power of n. In particular, blue $n = 1$, red $n = 2$, yellow $n = 4$. The dotted lines give the behaviors of the errors between the exact and the numerical solution for the different schemes. The theoretical orders are confirmed.

Partitioned Systems

Finally, we study the convergence of numerical schemes on a second-order differential equation, through the transformation of the equation in a partitioned system of equations. We consider the general IVP for a second-order differential equation:

$$\begin{cases} \ddot{y} = f(t, y) \\ \dot{y}(0) = a \\ y(0) = b. \end{cases} \tag{4.44}$$

If $f(t, y)$ does not directly depend on $\dot{y}$, we can transform the equation in a partitioned system of equations with the change of variables $\dot{y} = z$. Hence, we get the following partitioned system:

$$\begin{cases} \dot{z} = f(t, y) \\ \dot{y} = z \\ z(0) = a \\ y(0) = b. \end{cases} \tag{4.45}$$

In particular, we choose the setting $f(t, y) = \cos(t) + y$ and we get:

$$\begin{cases} \dot{z} = \cos(t) + y \quad t \in \langle 0, 1 \rangle \\ \dot{y} = z \\ z(0) = 0 \\ y(0) = 0. \end{cases} \tag{4.46}$$

With this choice, the exact solution reads:

$$\begin{aligned} y(t) &= \frac{e^t + e^{-t}}{4} - \frac{\cos(t)}{2}, \\ z(t) &= \frac{e^t - e^{-t}}{4} + \frac{\sin(t)}{2}. \end{aligned} \tag{4.47}$$

For this partitioned system, we compare the convergence behavior of the following methods:

- Partitioned Euler
- Partitioned Euler *à la partitioned Runge Kutta*
- Störmer-Verlet
- Störmer-Verlet *à la partitioned Runge Kutta*

We adopt the same techniques that we applied for the general systems in the previous paragraph. Initially, we compare the results with a coarse $\tau = 0.1$ for the standard Partitioned Euler Method and the Störmer-Verlet method in 4.3. It is straightforward to notice that the Störmer-Verlet method gives better accuracy than the Partitioned Euler Method. We finally study the convergence behavior: the outcome of the *Richardson method* is given in Figure 4.4 and in the following table.

Tau	eoc_PART	eoc_PART_RK	eoc_VER	eoc_VER_RK
0.1	1.0374	1.0374	1.9922	1.9993
0.05	1.0202	1.0202	1.9981	1.9998
0.025	1.0105	1.0105	1.9995	2.0000
0.0125	1.0053	1.0053	1.9999	2.0000
0.00625	1.0027	1.0027	2.0000	2.0000
0.003125	1.0014	1.0014	2.0000	2.0000
0.0015625				

Partitioned Euler Method is, as expected, of order 1, whereas Störmer-Verlet method is of order 2. In particular, the implementations of the standard and the Partitioned Euler Method and the *à la partitioned Runge Kutta* coincide, whereas the Verlet Method gives slightly different results between the two different implementations, but the convergence behavior is confirmed.

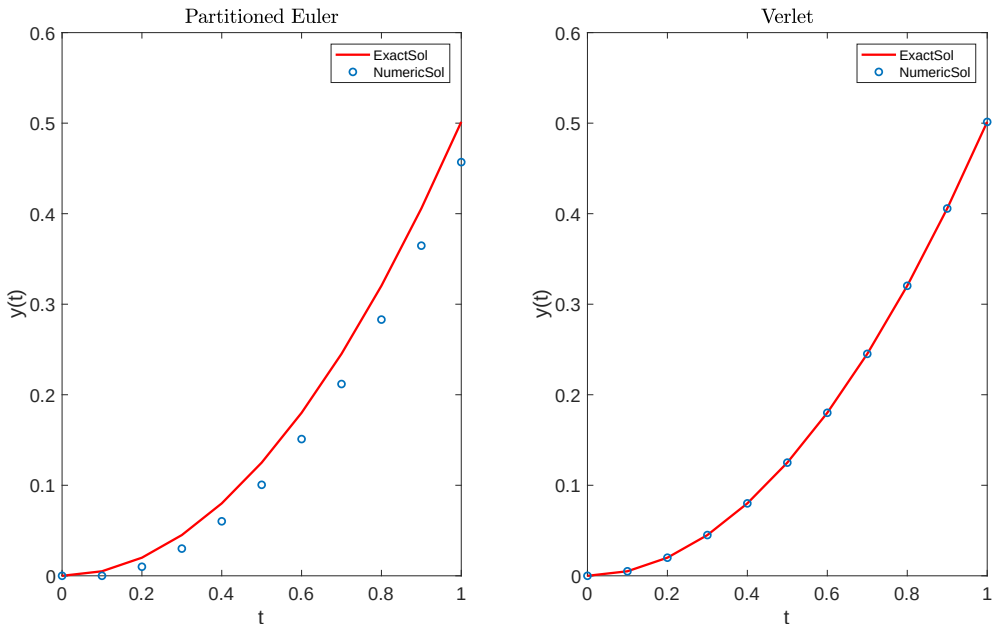


Figure 4.3: **Comparison between Partitioned Euler and Störmer Verlet** $\tau = 0.1$. The red straight line represents the exact solution, the dotted blue line the numerical solution. Accuracy is compared between the Partitioned Euler Method and Störmer Verlet. With a coarse $\tau = 0.1$, Störmer Verlet manages to approximate the solution better than Partitioned Euler.

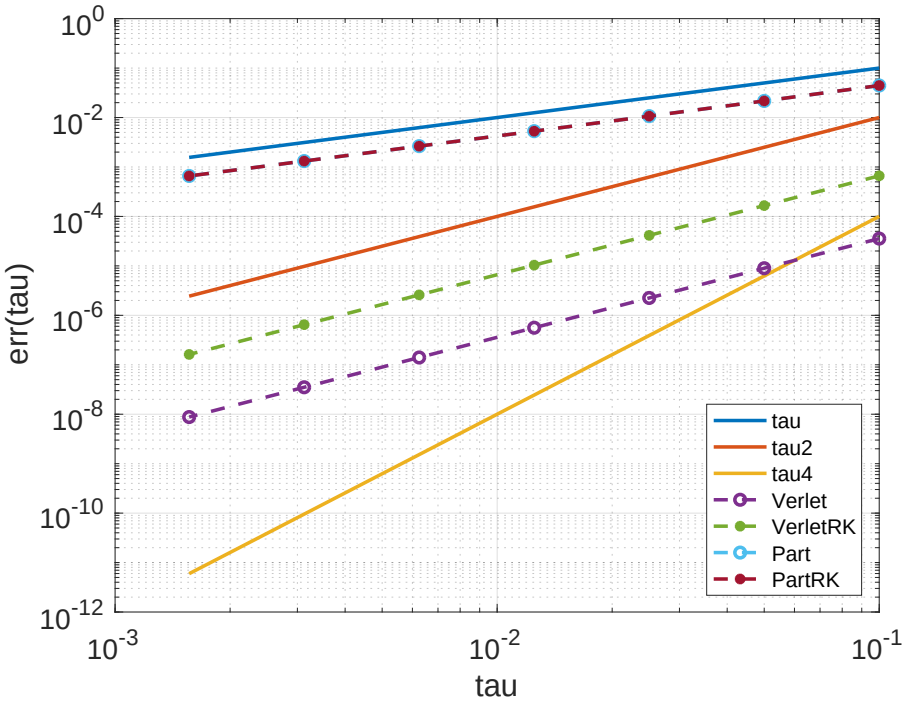


Figure 4.4: **Convergence of Numerical Methods for partitioned systems**. The straight lines represent the behaviour of the time step to the power of n. In particular, blue $n = 1$, red $n = 2$, yellow $n = 4$. The dotted lines give the behaviors of the errors between the exact and the numerical solution for the different schemes. The theoretical orders are confirmed.

4.5. Final Remarks

In this Chapter we analyzed some numerical approaches developed to solve general first order equations and partitioned systems, in particular Hamiltonian Systems. The emphasis was given to the Störmer-Verlet method and to its relevance into the set of numerical methods. In fact, a key feature of this scheme is to provide the connection between symplectic integrators, which are able to evolve in time preserving the geometrical structure of the problem, in particular the oriented projected areas, and Runge-Kutta methods, more precisely the partitioned Runge Kutta methods. The Störmer-Verlet method is characterized by order 2 of convergence, but more deep studies on partitioned symplectic Runge Kutta will give the opportunity to rise the order of convergence preserving the geometrical structure of the problem. The extensive usage of the Störmer-Verlet method in academic and industrial solver can therefore be explained by its capabilities of combining different properties, but it has to be considered as a starting point for more advanced and accurate numerical schemes.

5

OpenFOAM-HADES Files Coupling

The first stage of the MSc Project is to implement a coupling between OpenFOAM and HADES. To this purpose we decided to use the framework provided by CFDEM, a software package developed by DCS Computing, which allows fluid-particle simulations through the coupling between OpenFOAM and LIGGGHTS. The key idea is to use the structure provided by CFDEM and substitute the DEM solver, i.e. LIGGGHTS with HADES. A preliminary study of the CFDEM implementation is therefore necessary: we will investigate the data structures and the functions implemented in CFDEM. Before going in depth on the details of the implementation, It may be useful to describe very briefly the strategy adopted.

In Chapter 3 we discussed some constraints on the DEM and CFD time steps. Due to the different constraints, usually DEM time step is about 1-2 orders of magnitude smaller than CFD time step. The ideal case would be to adopt the minimum between the two and select it as global time step, but this would be extremely expensive from a computational point of view. Therefore, a strategy was developed to overcome this limitation and the result is a trade-off between efficiency and accuracy.

The idea is to couple the CFD and DEM phases every CFD time steps, but the CFDEM framework allows also the coupling as a multiple of ΔT_{CFD}. If the DEM time step is not fixed a priori but can vary over time (i.e. in HADES), it is required to save the data at least every CFD time step. But, in order to have an overview of the CFDEM implementation with LIGGGHTS as particle solver, we will initially consider the case of a fixed DEM time step. Hence, if ΔT_{DEM} is fixed and since it is smaller than ΔT_{CFD}, we select the CFD time step to be a multiple of the DEM time step, i.e. $\Delta T_{CFD} = k\Delta T_{DEM}$, where k is the *coupling interval* property. We will exchange information and perform the CFD evaluation every CFD time step, after k evaluation of DEM. This will introduce a *time delay* in the exchange of information: the particles will be subjected to a force which will be frozen during the entire DEM loop. We lose accuracy of the results, but we gain an extreme speed up of the simulation. In Figure 5.1 we provide a scheme which may be useful to visualize the strategy and to clarify the ideas behind the implementation.

In Section 5.1 we will describe the current CFDEM implementation. Then, in section 5.2 a list of the functions and files developed during the project is provided. Sections 5.4, 5.5, 5.6 describe in detail the several ideas and procedures adopted during the development.

5.1. Description of CFDEM framework

CFDEM is a C++ object-oriented library that was developed to perform fluid and particle simulations using OpenFOAM as CFD solver. The first numerical results in literature obtained via CFDEM are available in [16]. In this section, we describe the basic structure of CFDEM framework. More detailed information can be found in [9].

CFDEM basically substitutes and improves the Kinematic Cloud models implemented in OpenFOAM. To this purpose, the structure of CFDEM reflects the structure adopted in OpenFOAM. The functions implemented can be used to build *applications*, which can be *solvers* or *utilities*. Solvers perform the calculation to solve

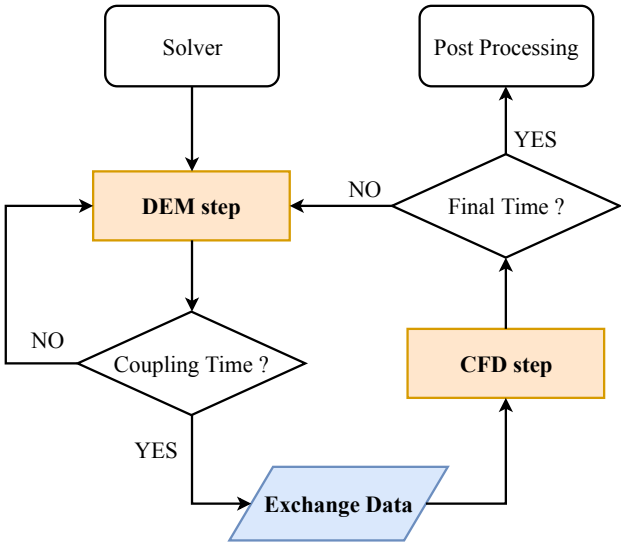


Figure 5.1: **Strategy for CFD and DEM coupling.** The orange boxes indicate blocks in which the computations are performed. In blue the phase of exchanging information is highlighted. The aim of this section is to study the implementation of this phase and to adapt it for the usage of HADES

a specific problem. Utilities prepare the mesh, set-up the simulation case and process the results. We now discuss briefly the applications available and the structure of the source code, which is organized in models and submodels.

Solvers and utilities

The solvers available in CFDEM are mainly:

- `cfdemSolverPiso`, for unresolved coupling.
- `cfdemSolverIB`, for resolved coupling.

Basically, they both are an implementation of the PISO algorithm, but in `cfdemSolverIB` an immersed boundary method is implemented in order to resolve the fluid at the particle level. The code is very similar to the implementation of PISO solver in OpenFOAM. The main difference is that inside the CFD loop, some functions of the CFDEM library are invoked. The main contribution is given by `particleCloud.evolve()`, which invokes the DEM solver and takes care of the exchange of data between CFD and DEM packages. In this case, particleCloud is an instance of the class `cfdemCloud` which will be described later.

Utilities present in CFDEM are mainly tools for post-processing of data.

Models and submodels

The source code of the library is organized in models and submodels.

The models implemented are the classes `cfdemCloud` and `cfdemCloudIB`, for unresolved and resolved coupling, respectively. They contain all the relevant data structures, implement the methods for the particle interactions (e.g. `cfdemCloud::evolve`) and they organize the different actions that are performed during a simulation by the submodels.

Various submodels are available and they perform precise tasks that are invoked during a simulation. Some examples of submodels are: dataExchangeModel, IOModel, ForceModel, voidFractionModel and so on.

File structure

The file structure needed to run a simulation reflects the structure already described for the usage of OpenFOAM. All the files are organized in the following hierarchical structure.

```
< case >
|- CFD
     |- 0
     |- constant
     ‘- system
‘- DEM
     ‘- post
```

Usually, in <case> a bash file, `Allrun.sh`, takes care of the execution of the commands needed by the simulation and the postprocessing. It may be useful to look at the file provided in the tutorials and adapt it to the specific case.

The `CFD` directory has the same contents of a usual OpenFOAM case. Note that in `constant` a file named `couplingProperties` provides the specifications of all the properties needed by CFDEM for the coupling. In the OpenFOAM terminology, this file provides a dictionary in which all the subModels are set for the current simulation: properties and settings are read by the constructors of the classes of the submodels and instances of these classes are created.

The `DEM` directory contains the input file of the DEM solver and the outcome of the simulation is available in the `post` directory depending on the settings provided by the input file to the DEM solver.

Outline of CFDEM simulations

We now briefly describe what is the general outline of a simulation using CFDEM framework and we select some important parts of the code that we will consider on the development of the new coupling.

The simulation outline reflects a typical OpenFOAM simulation. The user sets the computational mesh in the appropriate file and utilities are used to create the mesh. Usually the utility `blockMesh` is used, while `decomposePar` can be useful to set parallel computations. Then, the user runs a solver, which is one of the solvers of the CFDEM framework presented in a previous paragraph.

For the sake of simplicity, we consider the solver `cfdemSolverPiso` for the unresolved coupling. The resolved coupling works in a very similar way with the appropriate adaptations. Inside the solver, an instance of the class `cfdemCloud` (which we refer to as model) is created: the instance is named particleCloud. Inside a CFD loop, some functions of the CFDEM library are invoked. One of the most important is the call of `particleCloud.evolve()`. This calls the function `cfdemCloud::evolve` of the `cfdemCloud` class. This function is the responsible for the coupling and for the exchange of data between the DEM solver and CFD solver. Three important member functions are: `cfdemCloud::evolve`, `cfdemCloud::getDEMdata`, `cfdemCloud::giveDEMdata`.

When the instance of the class `cfdemCloud` is constructed, instances of classes of the various submodels are created. In particular, an instance of the class `dataExchangeModel` is created and named dataEchangeM. The class `dataExchangeModel` has three member functions which are relevant for the scope of this project:

- `dataExchangeModel::couple` runs the DEM simulation calling the DEM solver,
- `dataExchangeModel::getData` has the aim of transfer DEM data to CFD solver,
- `dataExchangeModel::giveData` transfers data from the CFD solver to the DEM solver.

In `cfdemCloud::evolve`, firstly the DEM solver is run via `dataExchangeM.couple()`, then data is get from the DEM solver through its member function `getDEMdata()`. This call invokes the function of the appropriate submodel: `dataExchangeM.getData()`. The same is done for giving data to the DEM solver for the next iteration. The procedure is described also by the scheme presented in Figure 5.2.

Then, after this process, the simulation goes back to the CFD level and the PISO algorithm is implemented in a very similar way to the standard OpenFOAM solver, except some minor changes to take into consideration the presence of the particles as an external force to the fluid, using the appropriate choice between the resolved and the unresolved approaches, according to the relative sizes between the particles and the CFD cells.

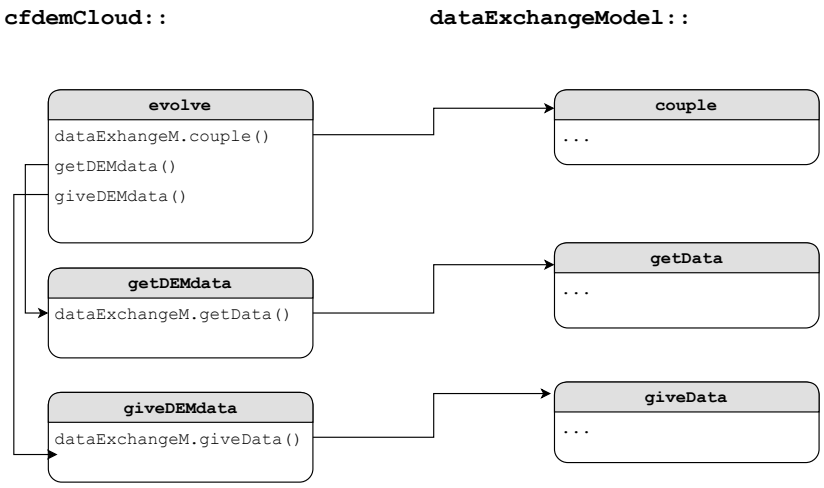


Figure 5.2: **Code structure for coupling and data Exchange.**

The `dataExchangeModel` submodel

The submodel that is responsible for the exchange of data between CFDEM and the DEM solver (LIGGGHTS) is therefore `dataExchangeModel`, a virtual class which is the base for the following methods:

- **TwoWayMPI**
  It implements the influences of the particles on the fluid and vice versa. It exploits MPI as communications between softwares. It is efficient and faster than the other methods.
- **TwoWayFiles**
  It implements the influences of the particles on the fluid and vice versa too, but the communication is achieved through (text) files.
- **OneWayVTK**
  It implements the influence of particle motion on the fluid dynamics. The communication is performed by VTK files.

In order to have control on the debug phase, the choice is to implement firstly the coupling between OpenFOAM and HADES via files, adapting the submodels TwoWayFiles and OneWayVTK to the data structures of HADES. An implementation exploiting MPI for communications between OpenFOAM and HADES and between processors in a parallel version of HADES would be ideal and efficient, but its development is beyond the scope of the current project. The result of this process will be the creation of a new data exchange model: **TwoWayHADES**. In order to create a new dataExchangeModel, a preliminary and detailed study on the existing procedure has to be pursued.

We now list all the data that are exchanged in both directions, i.e. form CFD to DEM solver and vice-versa. It is important to notice that different data are transmitted in the case of resolved or unresolved coupling: in the resolved case of resolved we require more data.

Data DEM → CFD through `cfdemCloud::getDEMdata` (or `cfdemCloudIB::getDEMdata`)

Unresolved
- Explicit Force
  - `radius`
  - `x` (position)
  - `v` (velocity)
- Implicit Force
  - ... +
  - `dragAcc` (drag acceleration)

Resolved
- Explicit Force
  - `radius`
  - `x` (position)
  - `v` (velocity)
  - `omega` (angular velocity)
- Implicit Force
  - ... +
  - `dragAcc` (drag acceleration)

Data CFD → DEM through `cfdemCloud::giveDEMdata` (or `cfdemCloudIB::giveDEMdata`)

Unresolved
- Explicit Force
  - `dragforce` (actually this is total force)
- Implicit Force
  - `Ksl` (K_{sl} coefficient of forcing term)
  - `uf` (fluid velocity)

Resolved
- Explicit Force
  - `dragforce` (actually this is total force)
  - `hdtorque`
- Implicit Force
  - `hdtorque`
  - `Ksl` (K_{sl} coefficient of forcing term)
  - `uf` (fluid velocity)

As stated before, the exchange of data is achieved through files or MPI as communicators.

In OneWayVTK, OpenFOAM simply reads the outcome of DEM solver at a fixed interval. The files are usually stored in `<case>/DEM/post`, but the user can set another location through an appropriate statement in the file `<case>/DEM/post/couplingProperties`. LIGGGHTS is told to save the output in VTK files, which are read by the member function of `OneWayVTK::getDEMdata`.

In TwoWayFiles, OpenFOAM reads or writes a file for every physical property stated in the tables above. The files are located in `<case>/CFD/couplingFiles`.

In TwoWayMPI, MPI is used as communicator between OpenFOAM and LIGGGHTS. The data are exchanged through appropriate functions implemented natively in LIGGGHTS. The source code can be found in the directorty `<LIGGGHTS>/LIGGGHTS-PUBLIC/src`, whereas for more details on the implementation, refer to [10].

5.2. New contributions

In this section, a simple list of the functions implemented or modified is provided.

In HADES:

- New Force module: `CFDForceModel.cpp`, `CFDForceModel.h` (From Gravity Module)
- Creation of `HADES.cpp`, `HADES.h`, `HADES_call.cpp`, `HADES_call.h` (from `main.cpp`, to simplify usage as library).
- Modification of `VerletIntegrator.cpp` and `VerletIntegrator.cpp`

In CFDEM:

- New Data Exchange Model: `twoWayHADES.cpp` and `twoWayHADES.h`

Interface:

- `count_particle_each_group.h`
- `search_output.h`
- `unite_output.h`
- `unzip.h`
- `update_runtime_HADES_input.h`

Compilation:

- Updated: `files` and `options` files in cfdemParticle/Make

Procedure for Coupling

The outline of the procedure is described in Section 5.1. To use HADES as the DEM solver, a new dataExchangeModel (TwoWayHADES) has been developed. We will now give a brief description of the running of a simulation using HADES as DEM solver.

After the set up of a case, a solver is run. The solver will call the evolution of the particle cloud. The evolving function will: get data arising from DEM (`twoWayHADES::getData`), perform the evolution of particle motion in time (`twoWayHADES::couple`) and give back data to DEM solver in order to set the correct forces for the next iteration (`twoWayHADES::getData`).
Some post processing steps are required to extract the relevant data from HADES output, since the output files are created for group of particles. The post porcessing of data involves the extraction of relevant information from each output file and the creation of a file for every physical property (radius, position, velocity, angular velocity) in which all the particles are considered.

5.3. Compiling

In order to couple OpenFOAM for fluid dynamics and HADES for particle motion, the first step is to compile the two software into a single one. To this purpose, we start the development linking the two softwares with a minimum influence on each other.

We decide to use HADES as a library: OpenFOAM will use HADES classes and functions. HADES is based on jem and jive, two opensource softwares developed by Dynaflow Research Group. The softwares have a precise hierarchy, jem is the low-level platform, it is a generic C++ programming toolkit, a mix between the standard C++ library and the standard Java packages. Jive is developed upon jem and it is specifically aimed at running numerical simulations, often involving large data sets. With jive it is possible to develop applications and HADES is an example. We will give more details on these toolkits in Chapter 6.

OpenFOAM and HADES present some conflicts and overlaps in the name of header files and classes. To this purpose a careful usage of the namespace has been adopted (mostly on general classes as Vector, Matrix, Property, String, ...)

The compilation of the two softwares is done through a simple call of an HADES simulation thanks to the function `main_hades` in the file `HADES_call.cpp`. We give as input to this function two strings (the first is arbitrary, the second is the path of the input file that we want to run, which is chosen between the tutorial provided during the installation) and the HADES simulation simply starts.

We decided to use the OpenFOAM framework for the compilation. OpenFOAM reads compilation instructions from three different sources: general outlines for the compilation (compiler, flags for optimization levels, for shared objects, ...) are provided by `<OpenFOAM-directory>/wmake/rules`, whereas specific commands for compilation of top-level application or source code is provided by `<top-level>/Make/{files, options}`. We do not modify the general directives, but we need to add some lines in the compilation of the library `cfdemParticle`, which contains the model and submodels source code.

We firstly need to place and compile HADES as a library. To this scope, inside the directory `cfdemParticle`, we place a directory `Hades-coupling`. Here we compile jem, jive and HADES in a sequential way. Once HADES has been compiled, we modify the files in `cfdemParticle/Make/{files, options}`.

In `cfdemParticle/Make/files` we just add the path of the `.C` file of the new dataExchangeModel (twoWayHADES):

```
$(dataExchangeModels)/twoWayHADES/twoWayHADES.C
```

In `cfdemParticle/Make/options` instead it is necessary to add some lines in the sections

- `EXE_INC`: the path of the header files is added.
- `LIB_LIBS`: the path of the libraries and the names of the jem, jive, HADES libraries that we use are added.

With these modifications, the compilation of HADES as a library, which can be used inside OpenFOAM through the CFDEM framework, is achieved.

5.4. Development of CFDEM structure

As we explained in Section 5.1, the development of a new `dataExchangeModel` is necessary to pursue the coupling of HADES with OpenFOAM. We will name this model `twoWayHADES` and we will adapt the functions of the existing subModels to the case.

The subModel `twoWayHADES` consists mainly on the implementation of four functions.

1. Constructor of the class
   The user-defined properties set in the file `<case>/CFD/constant/couplingProperties` are read from the dictionary and stored in appropriate variables.
   The properties are:
   - **DEMts** : DEM time step
   - **couplingInterval** : how many DEM steps every CFD loop
     Since, in the current implementation, the CFD and DEM time steps are kept constant, this can be evaluated as $\frac{\Delta T_{CFD}}{\Delta T_{DEM}}$
   - **inputFilename** : name of the HADES input file
2. `twoWayHADES::couple`
   This function is responsible for the call of the DEM solver and some pre- and post-processing procedures.
   In order to obtain the simplest coupling procedure, the idea adopted was to restart HADES every CFD loop, updating the simulation time interval every time. This is of course not efficient from a computational point of view and in a later stage of the project will be modified to increase speed and efficiency of the fluid-particle simulation. The first approach was therefore to start the DEM solver at every time step, updating the input file, perform the particle simulation and then shutdown the DEM solver.

   To this purpose, some modifications were necessary.

   First of all, the particle injection is performed at time 0, with an *ad hoc* input file. In the following steps, the particles are read and imported in HADES. Two different input files are therefore necessary:
   - `<case>.pro` for time 0
   - `<case>.pro_restart` for all the other cases.

   As we said, the input file needs to be modified at every CFD time steps in order to consider the appropriate time interval for the simulation. This is done through `update_runtime_HADES_input.h`. In section 5.6 the procedure is briefly described. Through this function, the following data are updated every CFD loop in the input file:
   - `DEM_ts`
   - `time_start_DEM`
   - `time_end_DEM`

   This is far from being efficient, but it gives a strong control in the implementation phase. Then, the function calls the DEM solver, from the function `main_HADES` in HADES_call.cpp.

   Since HADES saves output files in group of particles, some post-processing procedures are necessary after every CFD loop. These capabilities will be described in section 5.6.
3. `twoWayHADES::getData`
   The data arising from the DEM solver are saved in OpenFOAM. The function reads the relevant physical properties from an appropriate file, one for each property.
   The files are `<case>/DEM/exchangefiles/{radius,displacement,velocity}`. In each file, the information of the number of particles is stored in the first line. The typical content of one of these files (e.g. velocity) is:

```
nParticle 3 START
1.57038e-05 0.00840644 -0.00156704 0
1.00184e-05 0.00698293 -0.00156425 1
4.59195e-05 0.01111530 -0.00150084 2
```

4. `twoWayHADES::giveData`
   The data arising from the CFD solver are saved to a file, which will be read by the DEM solver during the next CFD loop. In Section 5.5 we will implement a model in HADES that will be able to read this information. The data consist on a numerically-evaluated force that act on the particles. They are saved in `<case>/DEM/exchangefiles/dragforce` and the structure of the file is kept similar to the files from the getData function. An example may be:

```
nParticle 3 START
2.50132e-11 4.23590e-08 2.97172e-10
3.08609e-11 4.41215e-08 2.88256e-10
3.00241e-10 3.83231e-08 5.71221e-10
```

5.5. Improvements in HADES

In order to perform fluid-particles simulation through HADES, some changes on the software were required. We will now describe the main changes and explain why they were necessary.

Usage as library

First of all, HADES is required to be used as a library and not as an executable anymore. This requirement led to a minor modification of the source code. The file `main.cpp` was divided in four files to exploit the usage as classes and member functions. The declaration and the definition of the mainModule, the class that contains all the structures required an HADES simulation are now provided in `HADES.h` and `HADES.cpp`. A calling function used to run HADES inside a CFD loop is instead provided in `HADES_call.h` and `HADES_call.cpp`. The four files contain nothing more than the original code but this structure allows a more user-friendly usage of HADES as a library.

Set initial time

A very important feature that had to be implemented in the outlook of the fluid-particle coupling was the possibility to decide the start time of a particle simulation. In fact, since HADES was originally developed for particle simulations, every simulation just started at time 0. To this purpose the integrator `VerletIntegrator` needed to be modified to accept an initial time field from the input file.

We will now describe the main idea behind the procedure adopted.

Firstly, we consider the class constructor: `VerletIntegrator::VerletIntegrator`. The integrator reads the property from the input file.

```
Properties  myProps = props.getProps  ( myName_ );
```

Then the value in the input file is saved int the variable `initTime_` (of type double).

```
myProps.find ( initTime_,      TIME_INIT       );
```

Then, in the initialization phase, the global variables `Globdat::TIME` and `Globdat::OLD_TIME` are set with the value of the variable `initTime_`.

```
globdat.set ( Globdat::TIME,         initTime_ );
globdat.set ( Globdat::OLD_TIME,     initTime_ );
```

This is unfortunately not enough.
We now consider the member function `VerletIntegrator::integrate`. The adding of those lines was not sufficient, since inside the integrator, during the first iteration, another global time variable (of the namespace `HadesNames`) is created and set to 0. This implies that also the previously modified variable is set again to 0.

Therefore, we set the new variable and the previous one again with the value from `initTime_`.

```
if (timeStep == 0)
{
  globdat.set ( HadesNames::NEXT_TIME, initTime_  );
  globdat.set ( Globdat::TIME,  initTime_         );
  globdat.get ( t0,             Globdat::TIME     );
```

```
}
```

This double modification is necessary because the implementation of the algorithm presents an unusual structure. In fact, the first iteration of the integrator has the role of an initialization of the numerical integration. The actual step forward in time is performed from the second iteration. This is the reason why we need to modify again the global data.

Now, with these modifications, we are able to perform a simulation deciding the starting time. The final time can be instead decided through a variable in the input file. The controlModule can read the information through the property `runWhile`.

A new Force Model

In order to obtain the coupling between the fluid and the particle dynamics, it is essential to have, in HADES, a tool that receive the information of the forces that arise by the influence of the fluid on the particles. This tool is not implemented in the standard HADES, so it is necessary to develop a strategy to take those forces in consideration. The CFDEM framework evaluates numerically some forces that the user can decide to consider in the simulation. We will exploit these evaluations, but we need a tool in HADES that is able to read this information.

To this purpose, it is necessary to develop a new force model in HADES that can be activated in case of *4-way* (full) coupling. The strategy used was to take inspiration from an already implemented force model. The easiest force to read and understand is of course gravity, which is implemented as a Model through `GravityModel.cpp` and `GravityModel.h`. We therefore used these source code files as a basis for the CFD-arising force. We decided to name this new module as `CFDForceModel`. The structure of the model is kept equal to the gravity one: classes and member function are the same. The key change in the implementation is performed in `CFDForceModel.h` in the member function `GravityModel::evalForces_`. Instead of evaluating the gravity force acting on each particle multiplying the constant acceleration g by the mass of the specific particle, we read the numerical values directly from the output file of the CFD solver, evaluated during the previous CFD iteration.

The total force is saved by OpenFOAM in `<case>/DEM/exchangefiles/dragforce`, using the CFDEM framework. Note that the name `dragforce` is not perfectly accurate, since the values can consider also lift forces, but it is kept for historical reasons.

Therefore, in `CFDForceModel::evalForces_` we firstly open the file in which the CFD-arising force was saved:

```
std::string  filename = "../DEM/exchangefiles/dragforce";
const char * filename_char=filename.c_str();
std::ifstream* inputPtr;

inputPtr = new std::ifstream(filename_char);
std::string just_read = " ";
while(just_read.compare("START") != 0)
{
  *inputPtr >> just_read;   //read until we read "START"
}
double double_just_read;
```

Finally, we insert in the force vector the values from the `dragforce` file. This is done simply reading the information in a for cycle. The order of the particles in the file is preserved by the order in which the information on radius, position and velocity is saved. We will go more in depth on this procedure in section 5.6

```
for ( dim = 0; dim < N; dim++ )
{
  *inputPtr >> double_just_read;
  fb [foffset + dim] += double_just_read;
}
```

Thanks to this new force model, the particles motion can be influenced by the presence of the fluid. The implementation allows a little lag between the CFD and DEM evaluations: the particles will feel the CFD-arising force evaluated at the previous CFD time step.
This lag is necessary to avoid the rise of computational cost: as explained previously, DEM time-step is required to be very small, usually 1-2 orders of magnitude lower than the CFD time-step and the common

strategy is to perform the DEM step forward in time freezing the CFD loop for k DEM iterations. The value of k can be set in `<case>/DEM/couplingProperties`.

5.6. Infrastructures implemented

In order to use the CFDEM framework with HADES, some intermediate transformations from the raw outputs needed to be performed. These intermediate steps were: how to set the correct time in HADES, how to read GZip-formatted output data, how to evaluate on the run the number of particles and how to build on the run the problem-dependent data structure.

How to set DEM initial time, final time and time-step

Thanks to the improvements on HADES described in section 5.5 it is now possible to start an HADES simulation for an arbitrary time interval.

This interval changes, of course, at every CFD time step. The procedure adopted to transfer this information from OpenFOAM to HADES, without using any MPI procedure, was to update the HADES input file, at every CFD step.
In the input file, before the specification of HadesModule, some variables can be fixed. For an arbitrary simulation we decided to require the following three variables:

1. `DEM_ts`, which sets the DEM time-steps, if the property `fixedTimeStep` in the integrator is set to `true`.
2. `time_start_DEM`, which sets the initial time for DEM simulation. It has to be set to the value of the previous CFD time. The value is read by the integrator in `initialTime`.
3. `time_end_DEM`, which sets the final time for DEM simulation. We need this to be the current CFD time. The value is read by the controlModule through the property `runWhile`.

We decided to update these variable at every CFD step, at the start of the DEM phase.
To this purpose, we developed the function:

```
void update_runtime_HADES_input (const char* filename, std::string property, double new_value)
```

It is implemented in `update_runtime_HADES_input.h`. It takes as arguments the path of the input file, the string containing the property to change and the new value to fix. The property `DEM_ts` is updated only at the first CFD loop and kept constant in all the next, whereas the other two are changed at every CFD time step.

Unzip

HADES stores the output of the simulation in GZip files. Whereas OpenFOAM, with the CFDEM framework, requires ASCII files to read data. To this purpose, a function

```
unzip (const char* argv )
```

was included in the software. The implementation is an adaptation of a function presented in a tutorial of the Jem software package. Since HADES is built on Jem, we easily adapted this feature in HADES.

Run-time information

In order to exchange data between HADES and OpenFOAM, some data structures needed to be modified to have a correct data transmission.

HADES saves output data by groups, which can be clusters of particles or even single particles. We decided to implement some functions to extract the relevant data (radius, position and velocity) from multiple output files. In HADES, we save the data through this procedure:

1. In hadesModule, in saveStructure we set:

```
nrOfSaves    =  1;
saveTime     = "$(time_end_DEM)";
```

In this way, we save data only when the DEM simulation has reached the final time.

2. In the generator, in each group of particles that we want to save we set:

```
outputFile  = "<path-input>";
```

The path will be `../DEM/<nameOfInputFile>`, since we call HADES from the CFD directory.

The number of groups and the number of particles in each group are only known at runtime, therefore we implemented some functions to read this information. Firstly, in `search_output.h` we implemented a function that reads how output files are created by the case that HADES is running. This is done counting how many times the word `outputFile` appears in the HADES input file. Then, in `count_particle_each_group.h`, a function that counts how many particles there are in each group has been developed. This is achieved counting how many lines in one field (e.g. velocity, or time) exist.

This are data relevant to the creation of a united output file, in which data from all groups (and therefore output files) are gathered. One file per physical property is generated, i.e. one for radius, one for position and one for velocity. These files are saved in `<case>/DEM/exchangefiles`. These are the final data that will be given as input to OpenFOAM to take into consideration the influence of the particles on the fluid dynamics.

Handling of Input files

For the sake of simplicity, all particles are injected at time 0 with an *ad hoc* input file, i.e. `HADES-input.pro`. From all the following time steps, another input file is used an it is named `HADES-input.pro_restart`. This file does not create particles, but it reads the positions and velocities from the output files generated by the previous time steps, whereas material properties are kept in both files. The main difference in the files is that in `HADES-input.pro_restart`, in case of full coupling, we need to add the force model `CFDForce`, in order to be able to read the information that arise from CFD simulation into the DEM solver. A check in the function `dataExchangeModel::couple` decides which input file to consider: only at first CFD time step it will select `HADES-input.pro`, otherwise `HADES-input.pro_restart`.

New Fundamental Input parametrs in HADES

In order to perform a simulation with the new implementation of HADES, some new variables are required in the input file. We give here an incomplete input file, where we kept the important properties developed for the CFD-DEM coupling. Note that the numerical values of `DEM_ts`, `time_start_DEM`, `time_end_DEM` need to be set only in pure HADES cases, since if we use CFDEM with the new implementation, this data will be set in every CFD iteration.

```
//  General data, definitions etc..
DEM_ts = 0.000010;
time_start_DEM = 0.00000;
time_end_DEM = 0.000010;

//  Specification of Models in HadesModule
  hadesModule =
  {
    saveStructure =
    {
       nrOfSaves =  1;
       saveTime  = "$(time_end_DEM)";
    };
    integrator = "Verlet"
    {
      fixedTimeStep   =  true;
      initialTime     =  "$(time_start_DEM)";          // Start DEM simulation at correct time
      timeStep        =  "$(DEM_ts)";
      charLength      =  1.0e-01;
      groupNames      =  [ <GroupsParticles to integrate> ];
    };
    model =
    {
      type   = "Multi";
      models = [ <Usual Models>,
                 "CFDForce"                            // In <HADES-input.pro_restart>:
               ];                                      // Activate in case of full coupling
      <ParticleModel> = <appropriate generator>
      {
```

```
      <standard Properties>
      outputFile  = "../DEM/files/<nameOfFIle>.gz";
      inputFile   = "../DEM/files/<nameOfFIle>.gz";   // In <HADES-input.pro_restart>:
    }                                                 // Read pos and vel of particles
  }
}
controlModule =
{
  runWhile  = "(t <= $(time_end_DEM) )";            // End DEM simulation at CFD timestep
  <standard Properties>
};
```

5.7. Run a test: example

In this section we run a sample case for exploring the new coupling features between OpenFOAM and HADES. Some particles collide between each other in a rectangular box, with fluid inside. Interactions with the walls are not relevant in this case, due to the small simulation time.
We will compare results for different levels of coupling between the particle motion and the fluid dynamics:

- No coupling
- 1-way coupling + particles collisions (fluid on particles but not vice versa)
- 2-way coupling + particles collisions (or 4-way coupling)

We consider 6 spherical particles, of the same size, i.e. with radius of 0.0005 m. We divide the particles in 2 groups: rightGrains (5) and leftGrains (1). The difference between the groups lies in a different resolution of the approximation of the spherical shape. In fact, HADES always considers contacts between polyhedrons and the user can specify, in the input script, the quality of the approximation of spheres (or ellipsoids) into polyhedrons. In Figure 5.3 we give a representation of the initial configuration of the system of particles. The representation is obtained from the view module in HADES.

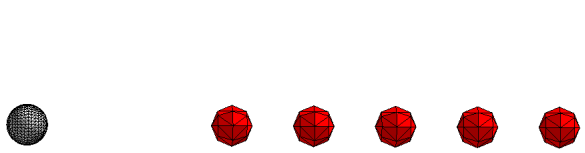


Figure 5.3: **Initial Configuration of the particle system**.

The particles of the group rightGrains are at rest, while we assign to the particle of the group leftGrains an initial velocity. This particle will cause the collisions. The initial configuration of positions and velocities of the particles are shown in Table 5.1, whereas the material properties are shown in Table 5.2.

	Position [m]			**Velocity** [m s^{-1}]			**Radius** [m]
	x	y	z	x	y	z	
RightGrains	0.005	0.005	0.050	0	0	0	0.0005
	0.005	0.005	0.048	0	0	0	0.0005
	0.005	0.005	0.046	0	0	0	0.0005
	0.005	0.005	0.044	0	0	0	0.0005
	0.005	0.005	0.042	0	0	0	0.0005
LeftGrains	0.005	0.005	0.055	0	0	-0.1	0.0005

Table 5.1: **Initial position, initial velocities and radius of particles**.

As far as the fluid is concerned, we consider a fluid at rest in a rectangular box. The domain has a square section (in x and y direction), with the edges of 0.01 m, whereas the extension in z direction is of 0.1 m. In Figure 5.4 we show the fluid domain and the orientation with respect to the axes. The initial values of the velocity and pressure are set to 0 in the entire domain. In Table 5.3 we set the physical properties that characterize the fluid dynamics. A visualization of the domain of computation is given in Figure 5.4. In Figure

Property	Value
Young Modulus	$E = 1 \cdot 10^2\ \mathrm{kg m^{-1} s^{-2}}$
Poisson Ratio	$\lambda = 0.3$
Friction Coeff	$\mu_{fr} = 0.7$
Rolling Friction Coeff	$\mu_{roll} = 4 \cdot 10^{-2}$
Normal damping Coeff	$\eta_n = 4 \cdot 10^2\ \mathrm{m^{-1} s^{-1}}$
Tangential damping Coeff	$\eta_t = 1 \cdot 10^2\ \mathrm{m^{-1} s^{-1}}$

Table 5.2: **Material Properties of particles**.

5.5 we provide the representation of the entire system (fluid and particles) at the start of the simulation time ($t = 0$).

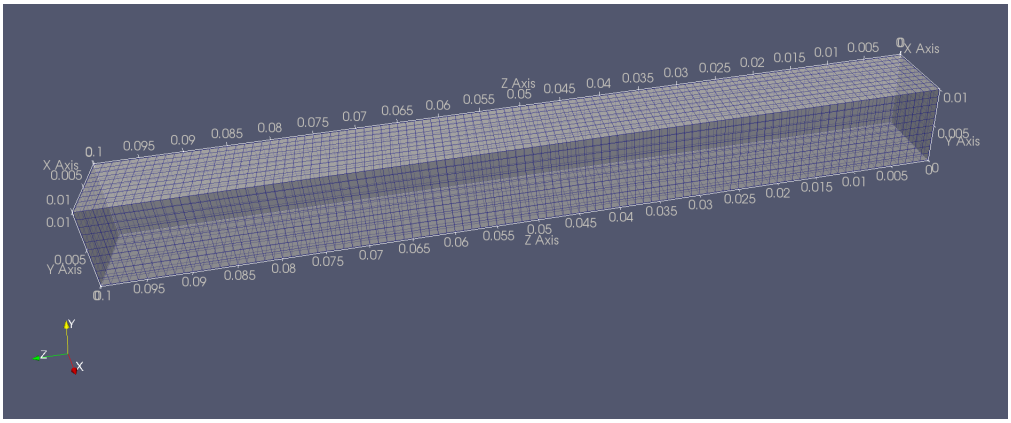


Figure 5.4: **Initial Configuration of the fluid**.

Property	Value
Density	$\rho = 10\mathrm{kg m^{-3}}$
Dynamic Viscosity	$\mu = 10^{-4}\ \mathrm{kg s^{-1} m^{-1}}$
Kinematic Viscosity	$\nu = 10^{-5}\mathrm{m^3 s^{-1}}$

Table 5.3: **Material Properties of fluid**.

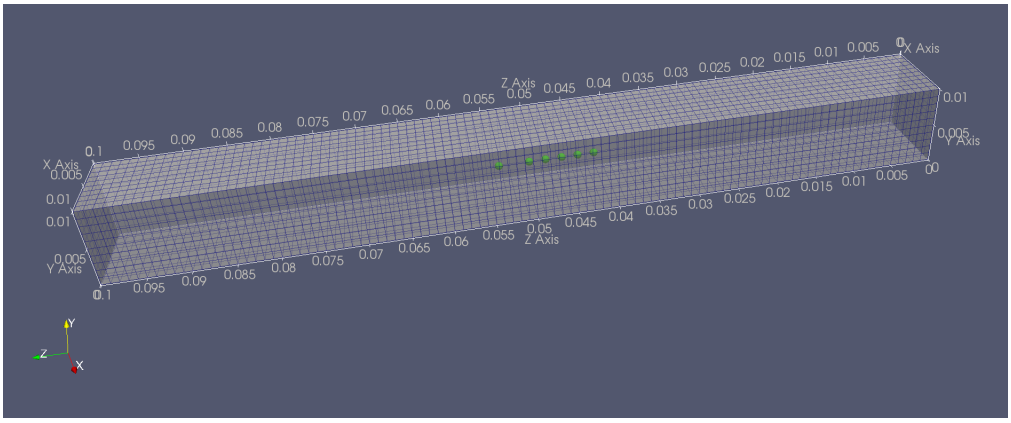


Figure 5.5: **Initial Configuration of the system: fluid and particles**.

We decide to simulate the dynamics in the interval $[0, 0.2]s$. To this purpose we impose the time steps and the

CFD-DEM coupling interval. The parameters are shown in Table 5.4

Property	Value
CFD time step	0.001 s
DEM time step	0.00001 s
Coupling Interval	100

Table 5.4: **Numerical parameters: time steps and coupling interval**.

Due to the relative dimensions of the particles and the fluid cell, we adopt an *unresolved* coupling. This is due to the fact that the resolved coupling is effective only when particles are much bigger than the computational grid, i.e. at least 8 CFD cells in a particle diameter.

We now list the forces that we consider in each level of coupling and provide some expectations on the outcome of the experiments.

It is important to notice that with the implementation provided it is not possible to have no coupling: the presence of the particles always influences the fluid dynamics, since in the CFD solver (`cfdemSolverPiso`) there is the call of the function

```
bool hasEvolved = particleCloud.evolve(voidfraction,Us,U);
```

Therefore, in order to not consider the particle motion influence into fluid dynamics, the "trick" is to avoid the solution of the fluid in time. This is possible through appropriate switches in the `couplingProperties` file. Thus, we can analyze the particle motion, which can be independent or influenced only by the initial condition of the fluid.

We remind that the exchange term that is inserted in the averaged Navier-Stokes equations is proportional to the relative velocity of the fluid and the particles.

$$R_{pf} = K_{pf}(\mathbf{u}_f - <\mathbf{u}_p>) = K_{pf}\mathbf{u}_f - K_{pf}<\mathbf{u}_p>, \tag{5.1}$$

where, in the last step, for numerical reasons, the expression is divided into implicit and an explicit term, using the cell-based averaged particle velocity $<\mathbf{u}_p>$. The exchange coefficient K_{pf} is modeled as:

$$K_{pf} = -\frac{|\sum_i F_{pf}|}{V_{cell}|(\mathbf{u}_f - <\mathbf{u}_p>)|}. \tag{5.2}$$

Therefore, the information will be updated at each CFD step, even if we do not solve the flow: the velocity of the particles changes and the exchange term in the averaged Navier-Stokes equation will be updated consequentially. Hence, not solving the flow is equivalent to couple the particle motions at each time step with the initial condition of the fluid.

We now describe the different levels of coupling that we explored, and in Figure 5.6 a visualization of the results is provided.

- **No coupling**:
  We do not impose fluid forces on the particle motion. We just consider collisions between particles, therefore we expect the motion of the particle to be entirely planar. We do not solve the fluid: it will continue to stay at rest over time.
- **1-way coupling**:
  We impose two fluid forces on the particle motion: Di Felice Drag and Archimede Lift. The forces are described in the paragraph below. Again, we consider the influence of the presence of the fluid on particle motion, therefore we do not activate the solution of the flow.
- **Full coupling**: We impose two fluid forces on the particle motion: Di Felice Drag and Archimede Lift. In this case, we solve the flow in time, in order to have the full coupling: forces on particles due to fluid, forces on fluid due to particle presence and particles collisions and interactions between each other.

Di Felice Drag

The drag force model is presented in detail in [48]. For the sake of completeness, we give here the model. Let $\mathbf{u}_i - \mathbf{v}_i$ be the relative velocity between the fluid and the particles i, ε_i the void fraction field in cell in which particle i lies and d_i the diameter of particle i. The the Di Felice drag model reads:

$$F_d = 0.125 C_{d0,i} \rho_f \pi d_i^2 \varepsilon_i^2 |\mathbf{u}_i - \mathbf{v}_i| (\mathbf{u}_i - \mathbf{v}_i) \epsilon_i^{-\chi}, \tag{5.3}$$

where:

$$\begin{array}{lcl} \text{Particle Reynolds Number} & Re_i & = \rho_f d_i \varepsilon_i |\mathbf{u}_i - \mathbf{v}_i| / \mu_f, \\ \text{Drag Coefficient} & C_{d0,i} & = (0.63 + 4.8/Re_i^{0.5})^2, \\ \text{Empirical Exponent} & \chi & = 3.7 - 0.65 \exp[-(1.5 - \log_{10} Re_i)^2/2]. \end{array} \tag{5.4}$$

Some error has been introduced considering the relative velocities: we see that equation (5.2) considers an averaged relative velocity, whereas equation (5.3). Details of this correction can be found in [17].

Archimede Lift

The Archimede Principle states that a particle immersed on a fluid received from the fluid a lift. In this case, the immersed volume is equal to the total volume of each particle. Let ρ_f be the density of the fluid and V the volume of a particle, the lift is then modeled as:

$$F_l = \rho_f g V. \tag{5.5}$$

Note that this definition of the buoyancy force does not take into account the gravity force that act on the particle. In order to have a correct physical behaviour, the gravity force needs to be added to the particle. Let ρ_p be the density of the particle then for a fully submerged object, with gravity acting in the direction of negative reference axis, the total force due to buoyancy and gravity is given by:

$$F = F_g + F_l \;\Rightarrow\; F = -\rho_p g V + \rho_f g V. \tag{5.6}$$

Comments on numerical results

In Figure 5.6 we provide a visualization of the numerical results: we plot 3D particles, whereas the CFD data is visualized in a slice of the domain that corresponds to the plane of the initial configuration of the particles.

For the cases of no coupling and one way coupling, in the slice we plot the void fraction field, since the velocity of the fluid is constant due to the fact that we are not solving the flow. Comparing the two cases, it is important to notice how the particle motion is planar in the case of no coupling, whereas it spans the third dimension in the case of one way coupling. Moreover, if we consider the projection of the particles on the z plane, we see that the particles in the case of one way coupling are slowed down: the collisions happen at later times. This is the tangible effect of the drag force. We remind that in the case of one way coupling, we are coupling the particle motion with the initial condition of the fluid, since the flow is not solved.

In the case of full coupling, we plot the velocity of the fluid in the slice of the domain. This is to highlight that the equation of fluid are now solved and that the fluid dynamics takes under consideration the presence of the particles. Comparing the full coupling with the one way coupling, we notice that the motion of particles is dramatically influenced by the fluid dynamics. The general motion of the system is slowed down and particles now are allowed to move also below the slice, since the property of the fluid are updated at each CFD time step and the forces the drag force depend on the relative velocity between the fluid and the particle.

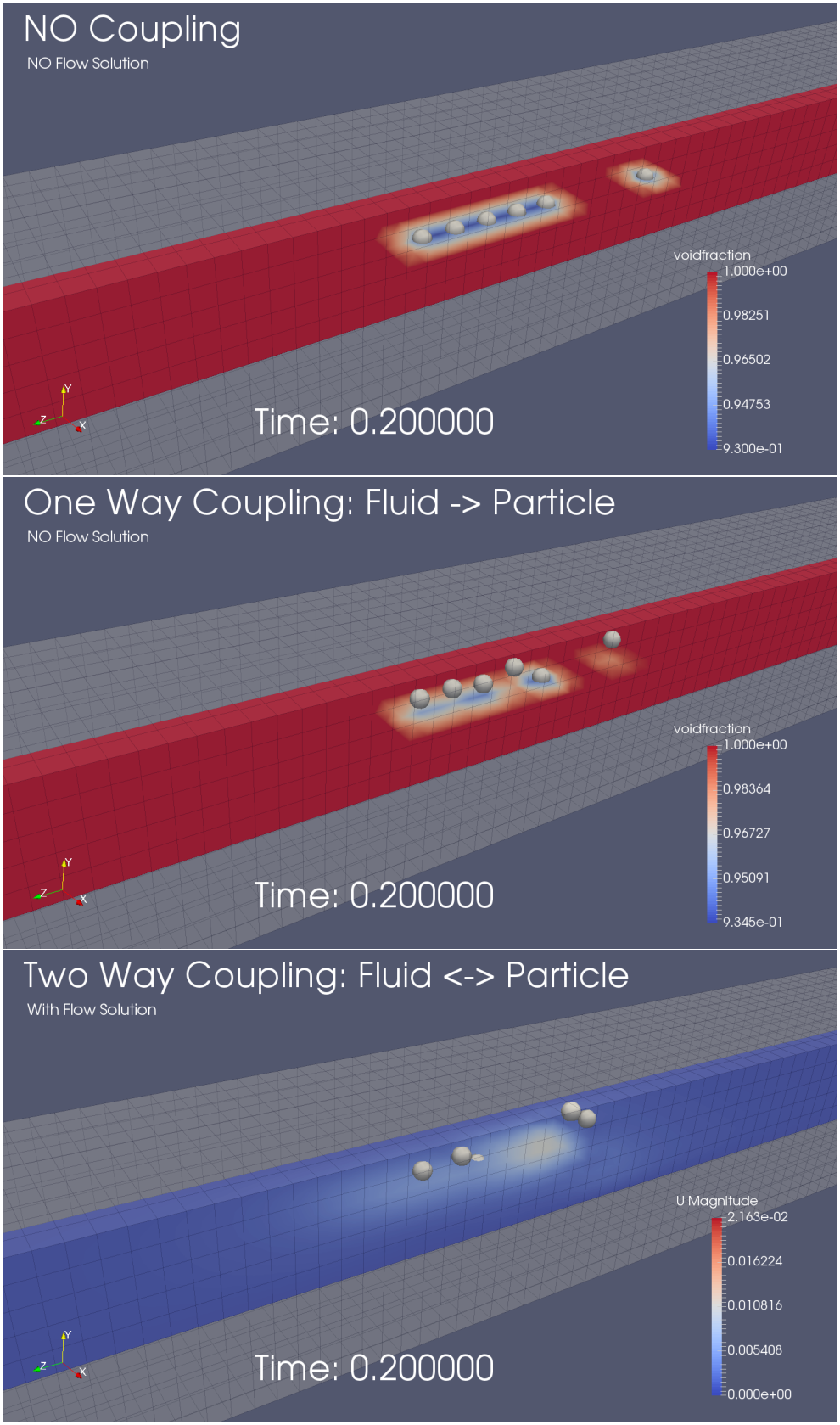


Figure 5.6: **Results for different levels of coupling**.

6

Improvements on OpenFOAM-HADES Coupling

In Chapter 5 we provided a complete description of a first rudimentary coupling between OpenFOAM and HADES for fluid-particle simulations. The approach was to adopt the simplest strategies in order to have a complete simulation working and we tested the implementation on a case for unresolved coupling.

We now focus on some limitations of the current implementation with the aim to improve the performance of the solver. We now provide a list and a brief description of the limitations.

The main drawbacks of the current implementation are:

- HADES is initialized every CFD time step. It performs the necessary calculations and then it is shut down. At the next CFD time step this loop is restarted.
- Two different input files are required, one for the first CFD step and another one for all the others.
- The CFD force is read at every DEM step inside every CFD loop, even if it is frozen. Since the file reading is really time-consuming, we should read this value just once per CFD step.
- The most important feature of HADES (not fixed DEM time step) is not exploited, since the DEM time step is decided *a priori*.

In this Chapter, we focus on these problems, we propose solutions and describe the new features of the implementation.

In Section 6.1 we focus on a coupling of HADES at a lower level in OpenFOAM. To this purpose, a description of the structure of HADES is necessary. We remind that HADES is based on Jem and Jive, two C++ softwares for numerical analysis. Hence, a discussion on Jem and Jive is given, especially on the structure of Modules and Models of applications built in Jive. In Section 6.2 we avoid the requirement of writing two different input files for the particle simulation. We add the CFD arising force on the run from the second CFD loop. In Section 6.3 we focus on the improvement of performances freezing the CFD arising force for the entire CFD loop, avoiding the reading of the force each DEM step. This will significantly speed up the simulation since it will allow for 1 evaluation from file instead of k, where k is the coupling interval property, as described in Chapter 5. Finally, in Section 6.4 we will allow a not-fixed time step for HADES, coupling CFD-DEM at fixed time and not every k DEM steps and in Section 6.5 we allow parallel CFD computations running HADES in a serial way.

6.1. HADES not restarted

In this Section we develop an implementation of the coupling of OpenFOAM and HADES which allows to initialize HADES and its data structure just once and not every CFD step inside the CFD loop.

Firstly, we provide some experiments where we prove why this step is necessary to have a speed up of the numerical simulation. For the moment, we will therefore focus only on pure DEM cases: we want to test how much the initialization phase can influence the performance of HADES. Hence, in the following, HADES is used *per se* and without the CFDEM framework. We consider a sample case and we perform the calculation up to different simulation times. Keeping the time step fixed, this is equivalent to perform the simulation for different numbers of DEM steps. We consider 100 (spherical) particles moving with a non-zero initial velocity and 1 particle moving in their direction to cause collisions. The ΔT_{DEM} is set to 10^{-5}. A file is saved at the end, saving the shape data of every particle and all the relevant properties (i.e. position, velocity, angular velocity, ...). In Figure 6.1 the execution time is plotted against the number of DEM steps evaluated.

It is straightforward to notice that the execution time is almost constant up to 100 DEM steps and then it increases drastically with respect to the number of steps. This proves that the initialization phase is extremely time consuming with respect to the actual time that HADES uses to perform a DEM step. We would expect, in fact, that the execution time would be always directly proportional to the simulation time (measured by the number of DEM time steps). Dividing the execution time of the case with maximum simulation time by the number of DEM steps, we have an estimation of the time required to perform a single time step: $2.43 \cdot 10^{-5}$. Unfortunately, the time required with a small number of steps appears to be much higher than (# DEM step $\cdot$ average time). A more detailed investigation is then required. We now want to investigate how much time is spent by HADES in the different phases of the execution. To this purpose, we fix the number of DEM time step and we evaluate for for each phase the time required by HADES to be completed. We compare the cases with and without saving the final file for the phases of initialization and running. In Figure 6.2 we plot the outcome of the experiment.

From the visualization of the results in Figure 6.2 we notice that, as expected, with this configuration of particles, the actual time required to perform one step forward in time is small compared to the other phases. The most time consuming phase appears to be the saving of the final files. This requires about 85.68 % of the total execution time. The initialization phase has large impact in the execution time: its contribution is about 13.07 % of the total. From these results, it appears obvious that improvements can be achieved operating on this two parts of the HADES implementation.

We decide to intervene on the initialization phase, in order to keep more control on debugging during the development of the project. However, a more direct communication will be essential for real-life industrial contexts. In the rest of this section we will firstly describe the structure of HADES and then we will describe a new implementation which allows to use HADES without restarting it every CFD loop.

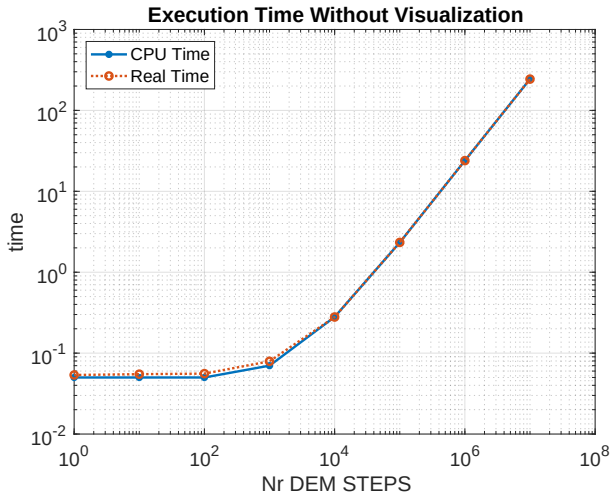


Figure 6.1: **Calculation time vs number DEM steps**. The calculation time is shown in dependence of the DEM steps performed. The expected direct proportional dependence between the number of steps performed is not respected for very small numbers of DEM steps. This highlights that the start-up time is significant in a pure HADES simulation. The data are obtained using 100 particles falling and 1 particle hitting this cluster, with no external forces applied and with $\Delta T_{DEM} = 0.00001$. An output file is saved at the end of the simulation.

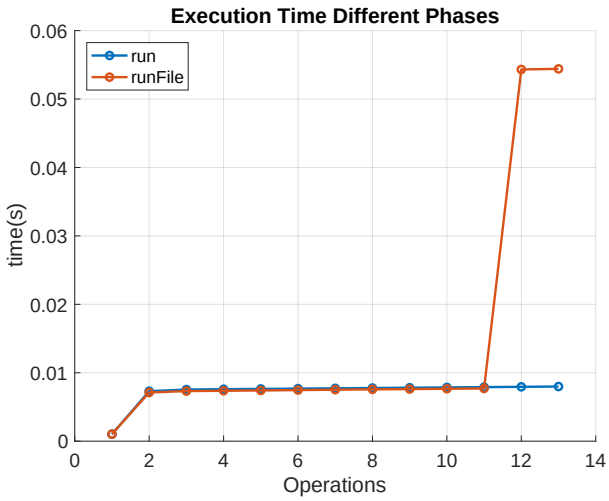


Figure 6.2: **Execution time of different phases in 10 DEM steps**. The visualization of the execution time of different phase is shown. The influence of saving a final output file in the execution time is explored. Operation 1 indicates the starting of the initialization phase, operation 2-11 deal with the DEM evaluation, operation 12 regards the saving/no-saving of the file, operation 13 is the shutdown phase. The same setting of previous plot is used, but we perform only 10 DEM steps. Two main contributions are evident: the first step (between operation 1-2) measures the time necessary for the initialization phase) and the step between 11-12 measures the saving time of the output file

6.1.1. Structure of HADES: Jem and Jive

HADES is a high level application which performs particle dynamics simulations. It is based on two C++ toolkits, named Jem and Jive, developed by Dynaflow Research Group. In the manuals of the two softwares it is reported that:

- Jem is a low-level generic programming C++ toolkit. It provides a portable interface to system-level services and serves as the foundation for Jive.
- Jive is specifically aimed at running numerical simulations, often involving large data sets.

The hierarchical structure of the toolkits allows a well-defined distinction between the levels of programming and a broad set of utilities are available to be used to built a user-specific application.

In Jem, classes are available to handle multi-dimensional arrays, manage memory, share data, perform I/O operations.
In Jive, classes are available to store and handle (unstructured) meshes/grids, assemble and solve (large) systems of equations, handling constraints and boundary conditions.

In the implementation of HADES, classes of both toolkits are extensively used and, in particular, the structure of Modules and Models provided by Jive is adopted. This structure allows to organize the implementation in a logical structure, exploiting the object-orienting features of C++.

6.1.2. Jive Modules and Models

As stated before, the aim of Jive is to provide an extremely flexible toolkit for developing numerical simulations. To this purpose, it uses a precise structure of Modules and Models, to provide an efficient decoupling between equations and algorithms to solve them.

It may be useful to develop algorithms which can solve different equations and to solve the same equations with different algorithms. This is the main idea behind the distinctions between Module (algorithms) and Models (equations). For example, in HADES, the different contact forces that can arise from a physical problem are implemented as Models, which can be activated by the user in the input file. Instead, integrators (Verlet) are implemented as Modules, since they can be used to solve different equations arising from the Models.

With this distinction in mind, Modules are generalized to take care of general operations, as setting the graphical visualization of the simulation (viewModule), printing information during run time (infoModule) or deciding until when running the simulation (controlModule). In HADES, all these Modules are gathered together in a ChainModule, called HadesModule. For the purpose of a more efficient implementation of HADES, we need to go a little more in depth in the concept of Modules.

Every application is built as a chain of modules and each module is an instance of the class `Module`, which is coded in the app package of Jive. The class Module has three member functions: `init`, `run`, `shutdown`, which are named after the different phases of simulation which they take care of.
The `init` function is called just once, at the start of a program. It is responsible of setting the properties and the configurations that the user sets in the input file.
The `run` function is called multiple times inside a program. Usually it is called in a loop, until some conditions are satisfied.
Finally, the `shutdown` function is called at the end if the program, hence just once.

The three functions receive as input parameters three instances of the class Properties: `globdat`, which contains the global database of the program, `props`, which contains the runtime parameters and `conf` that extends the properties in `props` using default values for the properties that are not set.

For example, after having declared a module MOD, as instance of the class Module, and conf, props, globdat as instances of class Properties, the module MOD can be initialized, run and shut-downed through:

```
MOD -> init     ( conf, props, globdat );
MOD -> run      ( conf, props, globdat );
MOD -> shutdown ( conf, props, globdat );
```

What will actually be done by the module is of course implemented in the definition of the particular member functions of the specific module. The main idea of the implementation of the modules was presented. Now we have the ingredients to understand the skeleton of HADES implementation.

In Jive, `Application::exec`, a member function of the class `Application` of the app package, provides the possibility of handling the various phases automatically. HADES uses this method to be set and run: in `HADES_call.cpp` a simple call to this function, using as input the `HADESModule` is enough to take care of the execution of all the different phases of all the Modules implemented. This is extremely user-friendly and easy to read, but it does not provide enough freedom on the usage of HADES.

In fact, as stated in the previous section, there is the need to initialize HADES once, perform some evaluations and then give the control back to the CFD loop, waiting for the new data and the new CFD time step to proceed with the calculations. This ambitious target is obtained subdividing the implementation of the `Application::exec` function in different member functions at the CFD level. In the next paragraphs we will explain how we achieved this result.

6.1.3. Implementation

As stated previously, the run of HADES in the current implementation is obtained by calling the function in `HADES_call.h`, which contains the call to the member function `Application::exec`. This functions takes care on all the phases of a HADES run: initialization, run, shutdown. Hence, we have to subdivide the content of the function `Application::exec` directly in the `dataExchangeModel` class. We will therefore have more freedom in the control of an HADES simulation, paying some effort in a stronger implementation.

We now describe the member function `Application::exec`. Firstly, some variables are declared, e.g. an instance of the `Module` class, instances of the `Properties` class, like the global database, properties and configuration. The global database is initialized and then there is the call of a very important member function of the `Application` class: `Application::runLoop`. Here we find the control of the different phases of the modules, and in the specific case, the `mainModule` of HADES will be controlled.

We give here the most important part of the function `Application::runLoop`, for the sake of completeness:

```
Module::Status  stat;

phase = "configuration";
mod.configure ( props, globdat );
mod.getConfig ( conf,  globdat );
```

```
phase = "initialization";
stat  = mod.init ( conf, props, globdat );
phase = "runtime";
...
while ( stat == Module::OK && ! Utils_::hangup )
{
    stat = mod.run ( globdat );
}
...
phase = "shutdown";
mod.shutdown ( globdat );
```

As we see, the different phases of an HADES simulation are controlled through these lines. We therefore want to distribute them directly in the implementation of the `dataExchangeModel` at CFDEM level. We have to divide the phases in two different parts and implement them in two different member functions of the `twoWayHADES` class, which is a realization of the `dataExchangeModel` virtual class.

- Constructor of the `twoWayHADES` class
  We implement here the initialization of HADES. Hence, at the end of the constructor we add all the code that originally was in the function `Application::exec`, until the call to the member function that controls the loop of the simulation: `Application::runLoop`. Then, we add also some parts of the code that we gave in the previous example: we add the configuration and initialization phase, but we do not include the call to the run function of the Module mod. In this way, during the construction of the instance of the `dataExchangeModel` class, we initialize HADES. This is performed just once and therefore we achieve the result to avoid the restart of HADES at every CFD step.

- `twoWayHADES::couple`. In this function we substitute the call of the function `main_HADES` with the `while` that we gave in the example. We have to slightly modify the implementation since we do not want to run HADES until the final time of simulation, but just until the next CFD loop. At the CFD level, this information is contained in the variable `time_end_DEM` and it is, of course, updated every CFD step. Therefore after every performance of the `run` function of the main module of HADES, we read the new HADES time and we continue to perform the simulation until this time is equal to `time_end_DEM`. We check this condition adding a control in the `while` statement. When the condition is no longer satisfied, we give the control back at the CFD level and we continue the procedure until we start the new CFD step.
  Up to now, the shutdown phase is not performed.

With these modifications we achieved the important result to initialize HADES just once during the runtime of the fluid-particle simulation.

In the new implementation, every property is initialized just once. Therefore, we just need one input file. This allows us to solve the problem of having two different input files, one for the first time step and one for all the others. The drawback of this improvement is that, in the case of full coupling, during the first time step, HADES is considering the CFD arising force, but this information has not been evaluated yet: the CFD force arises from the second CFD step. Hence, we have to slightly modify the implementation of the `CFDForceModel` in HADES in such a way that it reads the information only from the second CFD step. This is achieved in the next section.

6.2. CFD Force from second CFD time step

In the new implementation of the coupling, just one input file is required. Therefore, in the case of full coupling, the user will create an instance of the `CFDForce` in the input file. This information will be read also during the first time step, but at this moment the CFD arising force at the previous time step is of course not known and it cannot be read. Therefore, it is necessary to modify the implementation in order to set this force to 0 during the first CFD step.

The idea is that the `CFDForceModel::evalForces_` responsible for the creation of the force arising from CFD (reading it from the exchange file) is aware of the actual time of the simulation. Then, if the time is less than the first coupling time, the force is set to 0, otherwise it is read form the exchange file.

This is pursed firstly adding the `globdat` as input to the member function:

```
void CFDForceModel::evalForces_( const Dim<N>   d,      const Properties&  globdat)
```

Then we need to set a variable which sets the first coupling time. In the case of fixed time step, this will be (`DEMTimeStep` · `couplingInterval`). We just set it to 0.001 for an example.

```
CFDforce = "CFDforce"
{
    groupNames    = [ "leftGrains", "rightGrain" ];
    firstCoupleTime = 0.001;
};
```

Finally, every DEM time step, during the evaluation of the force, we check whether we are in the first CFD iteration. If we are in the first iteration, we set the force to 0, otherwise we read it from file:

```
if (time_HADES_ <= firstCoupleTime_)
{
    ...
    fb [foffset + dim] += 0.0;
}
else
{
    ...
    fb [foffset + dim] += double_just_read;
}
```

In this way we solved the problem of the first reading of the CFD-arising force, when the data is not yet available.

6.3. CFD Force frozen inside a CFD loop

The current implementation reads the CFD force every DEM step, even if the force is actually frozen for the entire CFD step, i.e. we read the information from the exchange file k times, where k is the coupling interval ($\Delta t_{CFD} = k\Delta t_{DEM}$). The next step to speed up the run time of the simulation will be to read the force just once in every CFD step and keeping it frozen for k DEM steps.

This is pursued through adding appropriate checks to the procedure described in the previous paragraph. Inside one coupling loop, it is important to distinguish due different moments: the reading (and saving in a vector) of the updated force at the first DEM step and the application of this force in all the successive iterations. To this purpose, since the information available inside the member function is limited, we adopt the usage of flags to identify the proper moment to set and reset the default values of these flags.

In particular, we have to organize the procedure in 3 different cases:

- **Start Case**: We set the flags to read the force from the fluid, store it in a vector and read it by the force evaluation
- **Mid Case**: We set the flags to read the force vector by the force evaluation
- **End Case**: We reset the default value of flags, in order to be ready to perform the cycle again in the next coupling loop.

Firstly, we created one attribute of the CFDForce class that could be modified by the member function developed for the force evaluation. This variable (`prev_iter_`) stores the number of CFD-DEM coupling steps performed up to that moment (at the start of the simulation it is initialized to 0) and it is updated at the END phase of each coupling loop. The Boolean variable `last_` is set to true at the END phase to describe the fact that we reached the end of a loop, therefore the next iteration would require the reading of the updated force.

Then, we evaluate the current iteration number: `iter` = `time_HADES_ / firstCoupleTime_`. Reading the value of the `last_` variable and comparing `prev_iter_` with `iter` we can identify the correct phase:

```
if (last_ == true)
{
  START CASE: set proper flags...
}
if (floor(iter) == prev_iter_)
{
  MID CASE: set proper flags...
}
else
{
  END CASE: update prev_iter_, set proper flags, ...
}
```

6.4. Not fixed DEM time step

As stated previously, the best feature provided by HADES is the possibility to integrate the particle motion equations with a not fixed time step. This is possible due to constraints on maximal displacements possible during an integration step. Allowing the implementation of the coupling to exploit this feature is fundamental and it can improve dramatically the performance of the solver. Thanks to the improvements described in this Section, the modifications necessary to achieve this scope are very limited.

In particular, to allow a not-fixed ΔT_{DEM} it is necessary to set the following parameters:

1. In CFD side: (`<case>/CFD/constant/couplingProperties`)

```
couplingInterval 1
```

2. In DEM input file (`<case>/DEM/<input.pro>`).

```
time_end    = ...;
CFD_ts      = ...;

hadesModule =
{
    saveStructure =
    {
        nrOfSaves = "floor($(time_end)/$(CFD_ts)) + 1";
        saveTime  = "$(CFD_ts)";
    };
    integrator = "Verlet"
    {
        fixedTimeStep = false;
        ...
    };
};
```

   We remind that since the coupling is pursued via files, the crucial phase in the coupling is the saving of files at the appropriate time. The information from the fluid to the particles is saved correctly through CFDEM, whereas the saving of information from the particle to the fluid needs to be tuned, since HADES has no clue that we need to save files every ΔT_{CFD}. This is the reason why it is necessary to set the parameters in dependence of the final time steps and the CFD time steps chosen in the simulation.

In order to have an appropriate tuning between the two softwares, a further check was introduced in the CFDEM submodel that allows the exchange of information, to avoid situations in which a different handling of digits between OpenFOAM and HADES could introduce further lags in the exchange of data. In fact, to strengthen the synchronicity between the particle evaluations and fluid calculations, the following checks were introduced to the code described in Section 6.1:

```
while (    (stat_ == Module::OK)
        && (abs(time_HADES - time_end_DEM) != 0)
        && (time_HADES < time_end_DEM)           )
{
    Info « "-- Start HADES Integration --" « endl;
    stat_ = HADES_private->run ( globdat );
    Info « "-- End HADES Integration --" « endl;
}
```

6.5. Allow CFD parallelization

Fluid-particle simulation of real-life situations often require extensive computational resources. In order to achieve faster results, parallelization of the code is extremely important. OpenFOAM provides native support to parallel computations, but as stated before, HADES is characterized by a serial implementation. The parallelization of HADES will be necessary to obtain fast results in industrial applications, especially in cases with a large number of particles involved, but it is not compatible with the framework of the current MSc project.

Hence, the strategy adopted was to allow parallel CFD computations while keeping the particle simulation serial. In particular, OpenFOAM adopts a master-slave strategy for the parallelization and we decided to keep this framework running HADES only in the master processor. Therefore, in the master processor we will have CFD evaluations of the appropriate portion of the domain and the DEM calculations, whereas in

all the slave processors only CFD evaluations take place. Since we are using files to communicate between OpenFOAM and HADES, the critical phases are the reading and writing of files. To this purpose, we used MPI (Message Passing Interface) to handle the communications between processors. In particular, we used basic MPI commands to send, receive and set barriers to achieve an appropriate communication, avoiding errors in the handling of files: in the phase of reading and writing, the processors open and close files one at a time, in order to maintain the correct data structure and order. Details on the code implemented are not relevant to this Thesis report, since they are easily readable from source code.

7

Non sphericity

The Discrete Element Method (DEM) is a powerful tool to achieve accurate results in particle simulations. The numerical results are particularly advanced in the case of spherical particles: various and precise models are available in order to detect and simulate contacts and several arising forces that depend on the physics of the problem. Hence, the hypothesis of sphericity is often applied, since a relevant number of drag and lift models are available. Unfortunately, the constraint to consider only spherical cases causes a severe limitation on applications, at the academic level and especially in industrial environments. Shale oil extraction, red-cell dynamics in blood, pills sedimentation are just some examples on the non-sphericity of particles in petroleum, bio-medical and pharmaceutical engineering. Due to the arising of computational power in the recent years, huge efforts have been done to improve the accuracy of simulations and to propose reliable models and numerical methods.

In this Chapter we briefly describe two different approaches that can be adopted to rise the accuracy of particle simulation:

- Quadric and Superquadric representation of particles
- Multisphere approximation

Both of the methods have advantages and drawbacks. In Section 7.1 we will describe the idea behind both the approaches and discuss the current available implementations, especially in the softwares that we are using for the fluid-particle simulations, i.e. OpenFOAM and HADES. After an elaborate discussion, we choose to implement the multisphere approach in HADES. This will be reported in Section 7.2. We implement an algorithm to approximate an arbitrarily shaped object from an STL file to a set a spheres, which will be uses as input in HADES. In this Chapter we will therefore focus on the implementation of a method to integrate the motion of sets of particles as rigid bodies: we will work, therefore, at HADES level.

7.1. Description of Possible Approaches

In this section we discuss the two main approaches that are adopted in academic and industrial research to simulate non-spherical particles: the quadric (or superquadric) approach and the multisphere technique. Extensive studies have been carried out in the recent years on both methods. In particular, useful reviews on available methods to consider non-spherical particle in the DEM procedure can be found in [32] and [47].

7.1.1. Ellipsoids or Superellipsoids

Generalizations of spheres in quadrics and superquadrics have seen an increasing interest in the development of computer graphics and visualization techniques. Afterwards, researchers started to use the new tools for engineering purposes, analyzing the mechanical properties of these objects. Two main contributions to a structured discussion of these topics are provided by [33] for quadrics and by [5] for superquadrics.

A straightforward generalization of spheres can be obtained considering particles with a quadric surface. Since we have to consider finite-sized particles with closed surfaces, the unique case that is relevant to the purpose of particle simulations is given by ellipsoids. In particular, the implicit equation that describes an ellipsoid with the center at the origin of the Cartesian system and the ellipsoid axes that coincide with the coordinate axes is given by:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1, \tag{7.1}$$

where a,b,c are real numbers that are named *principal semi-axes*. Of course, in the case of $a=b=c$ we obtain a sphere. It is possible to parametrize the surface of ellipsoid using the following relation:

$$\begin{aligned} x &= a\cos(\theta)\cos(\varphi),\\ y &= b\cos(\theta)\sin(\varphi),\\ z &= c\sin(\theta), \end{aligned}$$

with $-\pi/2\le\theta\le\pi/2$ and $-\pi\le\varphi\le\pi/2$. We use this parametrization for visualization purposes: in Figure 7.1 we give an example of ellipsoids with different parameters. With ellipsoids a generalization of spheres has been obtained, but this is still not sufficient to describe the huge variety of complex shapes that are relevant in real life situations.

A step forward can be obtained with a further generalization of quadrics, i.e. superquadrics. Superquadrics can be obtained substituting the squaring operations with arbitrary powers. Again, we are not interested in all the varieties that are included in the superquadrics set, but we limit our study to the finite-size object with closed surface, i.e. superellipsoids. In [5] a comprehensive presentation of the complex set of superquadrics is provided.

The implicit equation that describes a superellipsoid with the center at the origin of the Cartesian system and the ellipsoid axes that coincide with the coordinate axes is given by:

$$\left(\left|\frac{x}{a}\right|^{n_2}+\left|\frac{y}{b}\right|^{n_2}\right)^{n_1/n_2}+\left|\frac{z}{c}\right|^{n_2}=1. \tag{7.2}$$

Again, a,b,c are real numbers that are named *principal semi-axes*, whereas we introduced two *blockiness parameters*, n_1 and n_2. Note that if we choose $n_1=n_2=2$ we obtain an ellipsoid, and if we add the condition $a=b=c$, we obtain again a sphere. It is possible to parametrize the surface of a superellipsoid using the following relations:

$$\begin{aligned} x &= a\,\mathrm{sgn}(\cos\theta)|\cos\theta|^{2/t}\,\mathrm{sgn}(\cos\varphi)|\cos\varphi|^{2/r},\\ y &= b\,\mathrm{sgn}(\cos\theta)|\cos\theta|^{2/t}\,\mathrm{sgn}(\sin\varphi)|\sin\varphi|^{2/r},\\ z &= c\,\mathrm{sgn}(\sin\theta)|\sin\theta|^{2/t}, \end{aligned}$$

with $-\pi/2\le\theta\le\pi/2$ and $-\pi\le\varphi\le\pi/2$.

Using the framework of superellipsoids it is possible to obtain various approximations of regular shapes: round-edges approximation of cubes, cylinders, boxes and ellipsoids can be easily built. In Figure 7.2 we give some examples of the possible usage of superellipsoids for approximating regular shaped objects, varying the input parameters (a,b,c,n_1,n_2). In [36] a detailed implementation of superellipsoids for pure DEM simulation in LIGGGHTS is given.

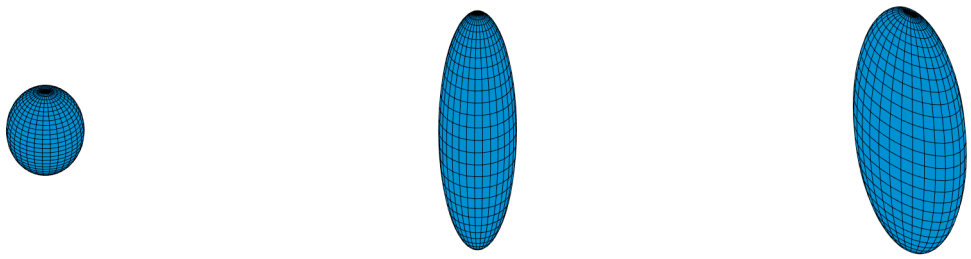


Figure 7.1: **Ellipsoids with varying parameters**. The first one is the ellipsoid generated with $a=b=c=1$, which is actually a sphere. The central object is obtained with $a=b=1, c=3$ and it is known in literature as prolate spheroid ($a=b\neq c$). The third one is a general ellipsoid ($a\neq b\neq c$), in the specific case $a=1, b=2, c=3$.

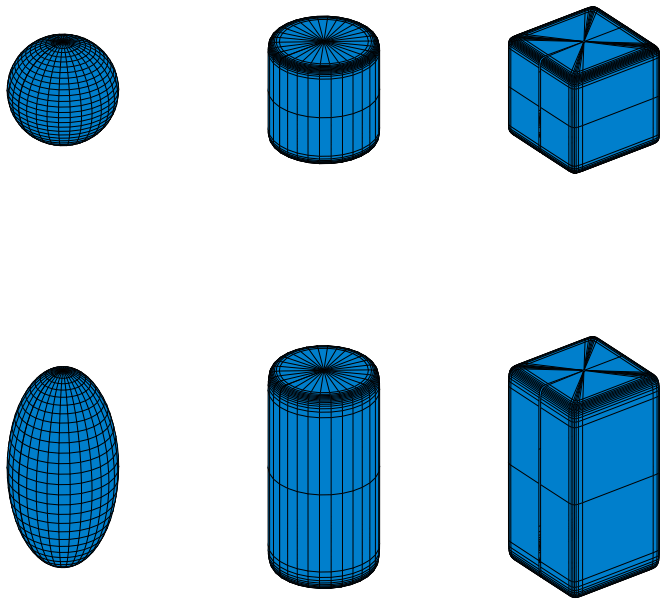


Figure 7.2: **Superellipsoids with varying parameters**. In the three figures at the top, $a = b = c = 1$, whereas in the three figure at the bottom, $a = b = 1, c = 2$. We see that we obtain sphere or ellipsoids if $n_1 = n_2 = 2$. If $n_1 \gg 2, n_2 = 2$ we obtain a rounded-edge cylinder. If $n_1 \gg 2, n_2 \gg 2$, we obtain a rounded-edge cube or box.

7.1.2. Multisphere

An alternative approach is to approximate an arbitrarily shaped object with a cluster of simple geometrical objects. Since spherical objects are easy to model as far as interactions, contact forces and contact detection are concerned, mostly clusters of spheres are used. The method is therefore named *multisphere approach*.

One of the key ideas of the approximation is that spheres are allowed to overlap to obtain complex shapes. For a complete description of an arbitrarily shaped object approximation, just a few data are required: only the positions of the centers of each sphere and their radii are required to fully describe an object. In order to use this approach, a pre-processing tool has to be developed: an algorithm that reads a random object as input (for example, via an STL file) and builds a cluster of spherical particles is in fact necessary.

7.1.3. Comparison between the methods and their applicability to DEM

Both superquadric and multisphere methods have been subjects of extensive studies. We now highlight the most important features of the different approaches.

Shape Approximation

Firstly, we need to consider the shape approximation of objects in the two methods. Superellipsoids allow excellent approximations of regular and symmetrical objects, but they provide inaccurate results if the objects are asymmetric, i.e. rocks.

Multispheres, instead, manage to approximate irregular shaped objects quite easily. The quality of approximation will depend on the algorithm used to build the cluster. Usually, approximation algorithms are particularly efficient on convex objects, but concave objects are poorly approximated also with superellipsoids.

Modeling of particle-particle interactions

We now consider the usage of the two approaches in DEM. As far as contact detection is concerned, algorithms have been developed for both methods. Contact detection for superellipsoids require solutions of

nonlinear systems and therefore it is computationally expensive. Usually, iterative methods are used, e.g. Newton's method. Moreover, convergence properties decrease with the increasing of blockiness parameters n_1 and n_2. On the other hand, the multisphere approach has the huge advantage that the same algorithm for contact detection for spherical spheres can be used. A summation of the forces that act on each particle of the cluster is performed and the torque arising from the translation of the forces to the center of mass of each cluster has to be evaluated. The main downside of the multisphere approach is that multiple contacts can occur, as reported in [27]. In this paper, a sphere is approximated with a multisphere cluster and performances of the method are analyzed. A contact between the big sphere and a wall is obtained form a physical experiment with different impact angles and the results are compared with numerical simulations of a sphere and a cluster of smaller spheres. The most important outcome of the experiment is that due to the freedom in the geometrical approximation of the object, results of the numerical simulation may differ from the experimental ones. In fact, depending on the geometrical distribution of spheres in the cluster, *multiple contacts* can occur in the cluster, while just one contact happens in the real experiment and in the numerical experiment with the standard spherical approach. The multiple contacts will cause an excessive stiffness in the impacts. Of course, in reality we will not have the possibility to compare the numerical results of the multisphere approximation with the arbitrary shaped object, since this would be too expensive form a computational point of view. The authors, therefore, propose a calibration of the physical parameters to take into account the possibility of the arise of multiple contacts.

Modeling of fluid-particle interaction

Finally, we need to consider the performances of the two approaches in the modeling of force arising from the influence of the fluid.

In order to achieve the coupling between the fluid and the particles, it is essential to locate the particles inside the CFD mesh. The two approaches handle this phase very differently. Using a superellipsoid approach, the location of a particle is very expensive from a computational point of view. The orientation of a particle is relevant and since we are not using spheres, a complete symmetricity is lost. In particular, the data required are: centers of the superellipsoid, the semiaxis and the orientation and *ad-hoc* algorithms have to be developed to build a correct voidfraction field. On the other hand, a multisphere approach just requires the center and the radius of each sphere and the structure available in CFDEM to locate pure spheres is ready to be used to locate each spheres of a cluster. This difference is valid both in resolved and unresolved couplings.

Instead, if we are using an unresolved approach, we have a further disctinction between the two approaches, since we require models for the drag and lift (forces or coefficients) to evaluate the forces of the fluids that act on each particle. This is not required in resolved couplings since we evaluate the integrated quantieties directly on the CFD mesh. As far as ellipsoids and superellipsoids are concerned, unfortunately a comprehensive study that covers all the shapes and the cases is lacking in literature. Lots of progresses have been done in the last decades, but results have been obtained for specific shapes and specific ranges of non-dimensional numbers, mainly of Reynolds number. In [46] models for the drag coefficient of regular shaped particles are proposed and in [47] a review of the currently available models for non-spherical drag coefficients are presented. In the second paper it is stated that sometimes superellipsoids are considered for contacts, but spherical drag and lift forces are used to describe the fluid forces. This is an example of how much work still needs to be done both on theoretical and on experimental sides for handling non-spherical particles. Using the multisphere model, instead, it is possible to use the models developed for single spherical particles for the drag and lift arising from the fluid influence and then sum up all the contributions due to each sphere in a cluster. Even if the accuracy may decrease in cases where the overlap between two neighbor spheres is consistent, this approach seems to be the best trade-off between accuracy and computational cost at the current computational power.

In Table 7.1 we sum up the different features of the two approaches.

Comments

In [40] a comparative study between the two methods has been carried out on shear tests and discharging of a flat bottom silo.

	Superellipsoids	**Multisphere**
Shape Approximation	regular, symmetrical objects	arbitrary objects
DEM interactions	ad-hoc detection algorithm ad-hoc model of contacts hugh compt. cost (1 object) cost constant per object	spherical detection algorithm spherical model of contacts low comp. cost (1 sphere) cost grows with # spheres tuning parameters required results depend on # spheres move clusters as rigid bodies
CFD-DEM Unresolved	drag-lift not well modeled exp. location in CFD mesh	drag-lift on each sphere cheap location in CFD mesh
CFD-DEM Resolved	exp. location in CFD mesh	cheap location in CFD mesh

Table 7.1: **Comparison between Superellipsoids and Multispheres approaches**.

In conclusion, with the current computational power, it has to be clear that accurate quantitative results for arbitrarily shaped objects are far from being achievable. Superellipsoidal particles and the multisphere approach are two methods that have been developed and used in academic and industries to fulfill the trade-off between accuracy and computational costs. There is not a unique, better method between the two approaches and the choice is, therefore, problem dependent. We can state that, neglecting the differences between symmetric and asymmetric objects, superellipsoidal particle are mainly the best methods for packing problems, whereas if evaluation of drag and lift is relevant for the problem, the multisphere approach can give more accurate results, since there is a lack in literature for models of fluid-arising forces in a general situation.

7.1.4. Current implementations

Both multisphere and superellipsoidal approaches are used in academic and industrial softwares to perform fluid-particle simulations of non spherical particles, but the set of implementations is far from being homogeneous.

As far as pure DEM computations are concerned, the two main softwares that we analyzed provide the following features:

- LIGGGHTS handles both superquadric and multisphere capabilities, but no algorithm to approximate an object with spheres is provided.
- HADES handles ellipsoidal particles and arbitrarily shaped object (via XML input files).

The two softwares cover a broad set of features, but the problem lies in their interface with the CFD solver. The key feature in HADES is the possibility of not fixing a DEM time step, since the algorithm automatically choose the best time step, not allowing too high displacements from one step to the next one. Exploiting this feature is the main point of the MSc project.

As stated before, the most important problem lies at the CFD level, i.e. in the location of the particles inside the CFD mesh. This problem is severe in the case of resolved coupling, where the shape of the body is extremely important for the evaluation of drag and lift on the object. The limitation becomes less restrictive in the case of unresolved coupling: since the particles are treated as points, the detection is not expensive, and the evaluation of the arising forces usually is performed applying to non spherical objects the same drag and lift models of spherical particles. To this purpose, details can be found in [47].

Therefore, the problems arise mainly for resolved coupling, where immersed boundary methods are applied to solve the fluid at particle level. Ideally, a complex object would be detected considering all the mesh points of its surface with Immersed boundary methods, but this process is definitely too expensive for the computational power that nowadays is available.

The actual implementations compatible with OpenFOAM are:

- Immersed Boundary Method provided by foam-extend4.0.
  It uses STL files to detect bodies inside the CFD mesh. Even if this provides reliable results, it is not useful to the scope of the project: at every coupling step the detection of a complex object has to be performed and the motion of an STL file has to be achieved. A simulation with a huge number of complex, non spherical particles is computationally not feasible.
- CFDEM implementation of Immersed Boundary Method.
  It provides an efficient framework for spherical particles: the information needed by the algorithm exploits the spherical shapes and therefore requires a really small amount of data: just centers of spheres and their radii are required.

In the current, open-source, implementation of CFDEM coupling with LIGGGHTS, only spherical particles simulation are possible.

7.2. Development of Multisphere

A resolved coupling simulation where every particle is described by an STL file which is read, translated and rotated at every coupling step is not feasible with the current computational power. Therefore, strategies have to be developed to achieve a trade-off between accuracy and computational resources.

The implementation of the superquadrics and multisphere approaches requires developments at different levels:

- Superquadric approach requires to develop a new technique to detect superquadric particles in the CFD mesh at CFDEM level.
- Multisphere approach requires to develop an algorithm to approximate complex objects with clusters of particles and to implement the motion of clusters of spheres as rigid bodies at HADES level.

Considering the state-of-the-art of the current implementations, the interests of industrial applications and the academic purposes, we decided to develop a multisphere procedure in HADES and exploit the spherical implementation of CFDEM to process the detection of spherical particles to achieve the coupling.

We now describe firstly an algorithm for the approximation of arbitrarily shaped objects with clusters of spheres and, finally, the implementation of a procedure in HADES to integrate the motion of the clusters treating them as rigid bodies.

7.2.1. Algorithm for Multisphere Approximation

In order to perform simulations of non spherical particles using the multisphere approach, it is essential to develop an algorithm to approximate an arbitrary shaped object with a cluster of spheres. To this purpose, we implement the algorithm presented in [3], where the authors proposed a general algorithm to approximate a generic object (convex or non-convex). Since visualization of intermediate results is important and helps the development of the algorithm, we decide to use octave as development platform.

The algorithm consists in the following phases:

1. Receive an arbitrary shaped object as input (as STL binary file)
2. Include the object in a Cartesian grid.
   The center points of the spheres will be some of the points of this grid
3. Evaluate the grid points that are inside and outside the object.
   Delete from the grid the outer points
4. For each point, evaluate the distances to all the triangles of the surface of the object file and store the minimum
5. Find the point with the maximum of the stored distances
   The point will be the center of the sphere, the distance the radius

6. Delete from the grid all the points that are at a fixed percentage of the radius far away from the center point of the new sphere.
   Repeat from 4. until the number of spheres added is reached.

7. Correction of volume (otherwise volume of cluster is always less of the volume of the object)

The first two steps are trivial and do not require a detailed discussion. Various open-source algorithms are available to read STL files in a wide amount of environments. Once the file is read, matrices are build to store vertices, faces and normals of each triangle of the surface of the object. Once the vertices are known, it is straightforward to build a Cartesian grid that includes the objects, just using some gap with respect to the minimum and maximum coordinates of the objects in the three directions. In Figure 7.3 we visualize the output of the first two steps of the algorithm.

We now discuss the third step. In order to classify the grid points in inside/outside points, the ray-shooting technique has been selected, an approach developed in 1968 by a IBM researcher. The idea is to select a reference point which lies outside the object and then send a ray from this reference point to each grid point. The ray is parametrized and through the solution of a 3×3 linear system it is possible, through a condition on the parameters, to evaluate if the ray intersects the surface of the object.

Let $\mathbf{p}$ be a point of the Cartesian grid, then we define a ray $\mathbf{r}$ as the parametrized line that starts from $\mathbf{p}$ with direction $\mathbf{d}$:

$$\mathbf{r} = \mathbf{p} + t\mathbf{d}. \tag{7.3}$$

The direction $\mathbf{d}$ is evaluated as the direction from the point $\mathbf{p}$ to the reference point that lies outside the Cartesian grid, whereas t is a parameter with the constraint $t \geq 0$.
Let now $\mathbf{a}, \mathbf{b}, \mathbf{c}$ be the vertices of a triangle, then in order to evaluate the intersection between the ray and the triangle, the following linear system needs to be solved:

$$\mathbf{p} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}), \tag{7.4}$$

where the 3 unknowns are the parameter t and the barycentric coordinates of the intersection β and γ. In matrix form the system to solve is written as:

$$\begin{bmatrix} -d_1 & b_1 - a_1 & c_1 - a_1 \\ -d_2 & b_2 - a_2 & c_2 - a_2 \\ -d_3 & b_3 - a_3 & c_3 - a_3 \end{bmatrix} \begin{bmatrix} t \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} p_1 - a_1 \\ p_2 - a_2 \\ p_3 - a_3 \end{bmatrix}. \tag{7.5}$$

An intersection occurs if $0 \leq \beta \leq 1$, $0 \leq \gamma \leq 1 - \beta$ and $t \geq 0$.

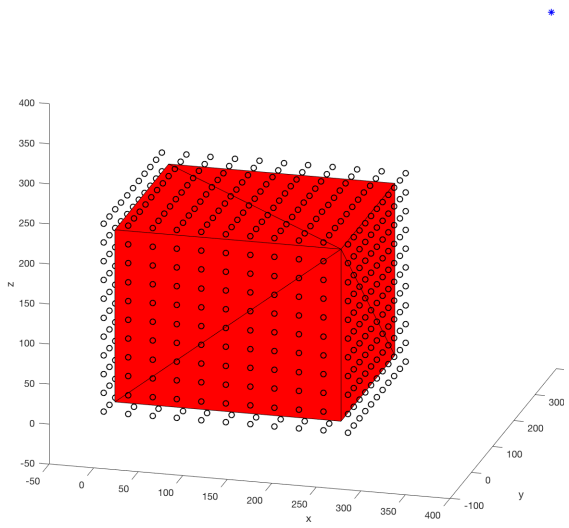


Figure 7.3: **Step 1. and 2. Grid generation**

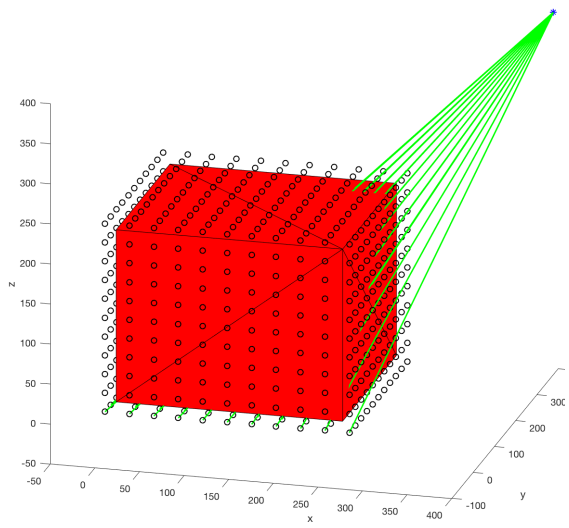


Figure 7.4: **Step 3. Ray Shooting**

In the case of multiple intersections, the nearest intersection with respect to the object surface is considered. Afterwards, the inner product between the normal of the surface and the ray direction is evaluated. If the inner product is negative, the point lies outside the object and it is removed form the grid. In Figure 7.4 we visualize the ray-shooting technique.

Once only the grid points inside the objects are kept, it is necessary to evaluate the distance between each point and each triangle of the surface. This is step 4. of the procedure. To this purpose, two possibilities exist: either all the points inside the triangle are at the same distance to the grid point, either the nearest point is a vertex or lies in the edge of the triangle. To distinguish these cases the grid point is projected onto the triangle surface. If the projection lies inside the triangle, the distance is evaluated simply as distance between the grid point and its projection, whereas if the projection lies outside, the distance is evaluated as the minimum distance between the point and each vertex or each edge, appropriately parametrized. This condition is checked again solving a 3×3 linear system. For each point, the minimum distance with respect to all the triangles of the surface is stored. We are now ready to insert the first sphere.

In step 5. we select the center of the sphere to insert and its radius simply finding the grid point with maximum distance from all the triangles. We evaluated all the minimum distances in the previous step so this is fast to evaluate. The maximum of the minimum distances is selected as the radius of the sphere.

Afterwards, in step 6. we remove from the grid some points that are contained in the new sphere. If no overlap between spheres is allowed, then all the points inside the new sphere are deleted. Since we would like to obtain overlaps, we select a percentage as input to the algorithm. The grid points with distances less than (percentage $\times$ radius) are deleted, the others are kept. In Figure 7.5 we visualize the creation of a sphere and the successive removal of grid points.

We repeat the procedure from step 4. until the number of spheres given as input is reached.

In step 7. we perform a volume correction of the cluster. In fact, with the algorithm described, the cluster will always approximate the object keeping all the spheres inside the original arbitrarily shaped objects. This implies that the volume of the cluster will always be less than the volume of the original object. To this purpose, we will multiply each radius of the cluster by a scale factor and, since we do not allow a shape deformation, we will translate the center of each sphere by a linear combination of itself and the center of mass of the object. Hence we need to calculate two quantities: the scale factor, as third-root of the volume of the object divided by the volume of the cluster, and the center of mass of the object.

Unfortunately, we cannot use any analytical formulas for the evaluation of the volumes, because the object is arbitrarily shaped and in the cluster overlaps between multiple spheres are allowed. Therefore, we adopt a *Monte Carlo method* to perform the volume evaluation. An implementation of the Monte Carlo method can be found in [37], and this was used as inspiration to the implementation inside the algorithm.

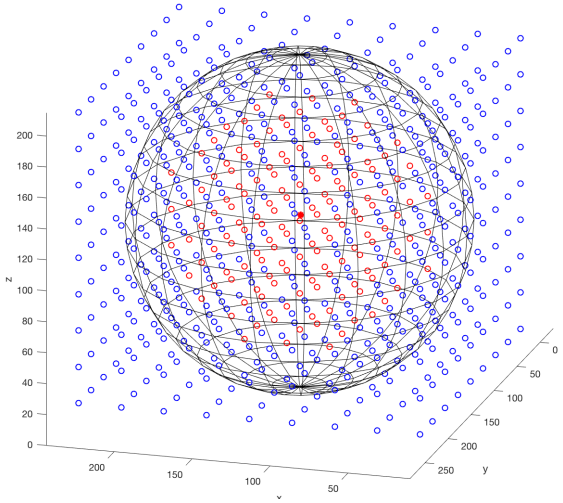


Figure 7.5: **Step 6. Deletion of points**

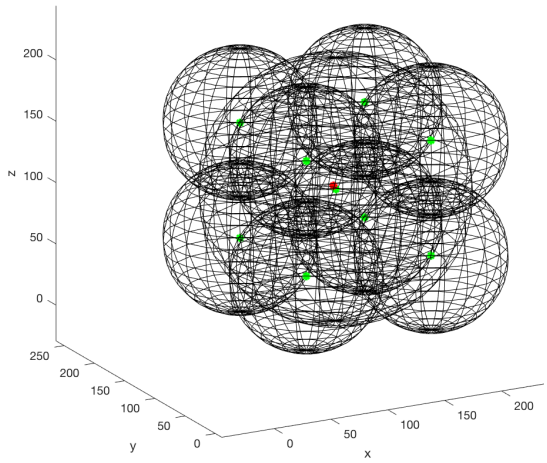


Figure 7.6: **Step 6. Partial cluster**

Monte Carlo methods are a set of computational methods developed to solve complex problems exploiting the generation of random or pseudo-random numbers to obtain numerical results. Applications of Monte Carlo methods can be found in various different contexts where numerical simulations are performed: fluid dynamics, financial engineering and signal processing are just some examples of the usage of these methods. The structure of Monte Carlo methods is given by the generation of random points with a fixed probability distribution in a computational domain, the execution of an algorithm with these random points and, finally, the evaluation of a global quantity. In our case, we apply a Monte Carlo method to perform the numerical integration of volumes and of the quantities necessary to evaluate the center of mass.

We define as domain the original grid that was built in step 2. of the algorithm and we sample N random points with a uniform distribution inside this domain. Due to the fact that the grid is Cartesian, this is done without particular effort. In order to evaluate the volume of the STL object, we perform step 3. for every point of the grid and we count how many points lie inside the object. The number of inside points divided by the number of total points and multiplied by the volume of the Cartesian grid that we built is an approximation of the volume of the object.

Formally speaking, the volume of the object is described by:

$$V_{obj} = \int_{\mathbb{R}^3} f_{obj}(\mathbf{p})\, d\mathbf{x}, \tag{7.6}$$

where $f_{obj}(\mathbf{p})$ is an indicator function, i.e. it takes value 1 if p lies inside the domain, in this case the object or 0 otherwise:

$$f_{obj}(\mathbf{p}) = \begin{cases} 1 & \mathbf{p} \in \text{Obj} \\ 0 & \mathbf{p} \notin \text{Obj} \end{cases} \tag{7.7}$$

Since we are using a uniform sampling, defining M as the number of points that lie inside the object and V_{box} the volume of the Cartesian grid generated in step 2., we obtain the approximation:

$$V_{obj} \approx \frac{M}{N} V_{box}. \tag{7.8}$$

The evaluation of the volume of the cluster is performed with the same method, but since overlaps are allowed, one should be careful to evaluate each grid point in the intersections just once.

In order to evaluate the center of mass of the object, we apply the constant density hypothesis of the object, therefore we need to find the centroid C of the object. Each components of the centroid is obtained as:

$$C_x = \frac{\int_{\mathbb{R}^3} x\,dx\,dy\,dz}{\int_{\mathbb{R}^3} dx\,dy\,dz} \quad C_y = \frac{\int_{\mathbb{R}^3} y\,dx\,dy\,dz}{\int_{\mathbb{R}^3} dx\,dy\,dz} \quad C_z = \frac{\int_{\mathbb{R}^3} z\,dx\,dy\,dz}{\int_{\mathbb{R}^3} dx\,dy\,dz}. \tag{7.9}$$

For the evaluation of the integrals we apply the same procedure that we adopt for the volume evaluation.

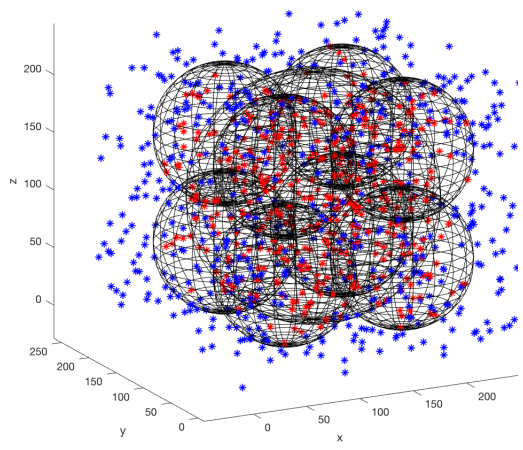


Figure 7.7: **Step 6. MonteCarlo Integration**

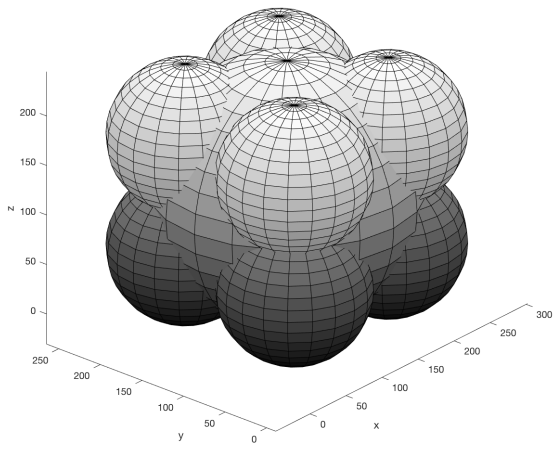


Figure 7.8: **Step 7. Final cluster**

We just discussed how a Monte Carlo method can be applied to evaluate volumes and centers of mass of arbitrarily shaped objects. This approach allows to have numerical results in complex situations, where analytical approaches fail due to irregular shapes. The numerical error produced by the Monte Carlo approaches decreases with $1/\sqrt{N}$, where N is the number of random points. Note that due to randomness of points, the error may be subjected to oscillations, but, due to the central limit theorem, the general behavior will decrease with the aforementioned ratio.

Computational cost

We briefly analyze the computational cost of this method for the approximation of complex objects with a cluster of spheres. The most expensive parts of the algorithm are the detections of inner points of complex objects, which are required in step 3. for the subdivision of grid points in inner and outer points and in step 7. to evaluate the volume of the object. The algorithm is the same, but the difference is that we are considering grid points in step 3. and uniformly distributed random points in step 7.

Let us consider step 3. Let n be the number of grid points ($\sqrt[3]{n}$ are grid points per edge of the Cartesian Grid) and T be the number of triangles of the surface of the STL object.
Then, the solution of $n \cdot T$ linear systems of 3×3 is performed to find intersections in the ray shooting technique ($\mathcal{O}(3^3)$). Afterwards, for every intersection (bounded by n) the evaluation of a minimum is performed ($3^3 = 27$ flops), followed by the evaluation of a norm of a 3 component vector. This last operation is negligible if $n, T \gg 1$. The same holds of step 7. where be consider m uniformly distributed random points.

Therefore, the approximated computational costs are:

- step 3 : $n \cdot T \cdot 27$ flops $+\mathcal{O}(n)$
- step 7 : $m \cdot T \cdot 27$ flops $+\mathcal{O}(m)$

It is straightforward to notice that the cost is increasing in the number of triangles and the number of grid points (or random points). Unfortunately, the number of grid points n has to be quite large to have accurate results and also the number of random points m required by the Monte Carlo method needs to be considerable, due to the convergence behavior $1/\sqrt{m}$.

If the surface of the object is characterized by an extremely fine mesh, it may be necessary to coarsen its surface before applying the algorithm, in order to avoid excessive calculations in the approximation phase. An example is given in Figure 7.11 where an STL file of a rock downloaded from a NASA dataset is considered. In order to avoid excessive calculations, an approximation of the rock has been performed through an OpenFOAM utility `surfaceCoarsen` and it is visible in orange. Afterwards, the algorithm has been applied to the *approximated rock* and the resulting cluster is plotted in blue.

Error of Approximation

After having obtained a cluster of spheres that approximates an arbitrarily shaped object, it is interesting to evaluate the errors that arise from the approximation and to analyze their dependencies with respect to the relevant parameters. In this case, the input parameters are the number of the spheres of the cluster and the overlap allowed. In particular, overlaps are always allowed, but the parameter involved is the percentage that is used in Step 6. In fact, grid points are removed at the distance of $percentage \cdot radius$. Therefore, in the case of $percentage = 1$, the center point of new spheres will lie outside the sphere, but the sphere are still allowed to overlap.

In literature, a unique procedure to compare approximation of objects by cluster of spheres is not available. We decide to use two distinct approaches: a simple relative error on the volumes and the *offset volume*, which is proposed in [3].

The first measure of error is given by:

$$err_{vol} = \frac{|V_{obj} - V_{cluster}|}{V_{obj}}. \tag{7.10}$$

Instead, the *offset error* is evaluated considering all the points that are wrongly included or excluded by the approximation. In fact, in the approximations, there exist some portions of the object which are not included

in the cluster and some portions of the cluster which are not included in the object. We evaluate these two contribution via a Monte Carlo procedure and we define their sum as the *offset volume* V_{off}. Hence, we have a second definition of an error:

$$err_{off} = \frac{V_{off}}{V_{obj}}. \tag{7.11}$$

In Figure 7.9 we analyze the behavior of the two different definitions adopted as errors, varying the two main parameters. In the left, we use the err_{vol} definition (7.10), whereas in the right we use the err_{off} definition (7.11).

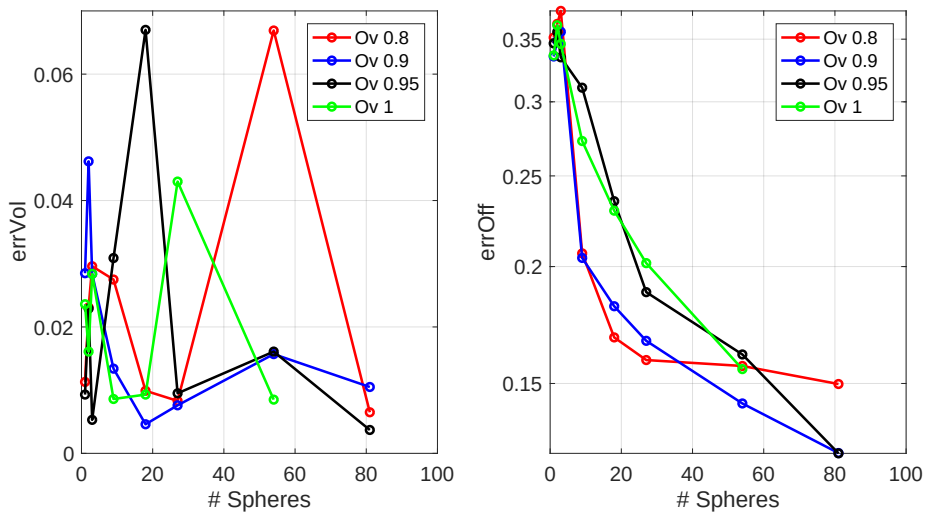


Figure 7.9: **Error of Approximation of a Box with a cluster of spheres as function of number of spheres.** The errors are shown varying the number of the spheres and the percentage chosen in Step 6. In order to delete from the Cartesian grid candidates for the centers of next spheres in the cluster. In the x axis, we use the number of the spheres, in the y axis we plot the value of the errors. In the left, err_{vol} is analyzed, in the right err_{off}.

It is straightforward to notice that using the definition of err_{vol}, strong oscillations appear and increasing the number of spheres in the cluster does not imply a decrease in the error. Nevertheless, it is important to state that the relative error, as far as volumes are concerned, is quite small. All the cases show an error in the interval of 1% - 6%, which leads to quite accurate results. Even if the errors are small, this definition of error does not allow a study of the convergence of results. A possible explanation of the presence of oscillations may be the attempt to approximate a symmetric object with a cluster that is not subjected to symmetric constraints. Different percentages of deletion of grid points lead, in fact, a structure that are not symmetric and this can be a cause of the rise of the error.
Instead, measuring errors with the *offset* definition leads to results in which correlation between the number of spheres and the behavior of the error is clear. The range of errors with this definition is between 12% and 35%, but the reader should not compare these high values with previous ones because the definitions measure different aspects. It is evident that increasing the number of spheres decreases the error. Some oscillations in the trend are easily noticeable and, again, symmetry reasons seem to be an explanations of these phenomena.

In both cases, Monte Carlo methods had to be implemented to evaluate the volumes, therefore statistical oscillation may play a role in the behavior of the error. To have a small influence of errors, a quite high number of trial points in the domain has been considered (10000). Remind that the convergence behavior of Monte Carlo methods is $1/\sqrt{N}$. Attempts to achieve a precise correlation between the number of spheres and the error have been made, but only a generic relation could be obtained. Further studies could be dedicated to this topic, in order to reach a deeper level on the prediction of errors in the approximation.

To conclude, it is important to state that, between the two approaches, the *offset* procedure proposed in [3] performs better in the analysis of the error behavior, if compared with a simple relative volume error evaluation. However, the reader has to keep in mind that, if a huge number of particles is considered in a DEM simulation, the approximation of each particle with a big number of sphere will be subjected to a strong

limitation by the computational power available. Therefore, a trade-off between accuracy and computational cost will be always required.

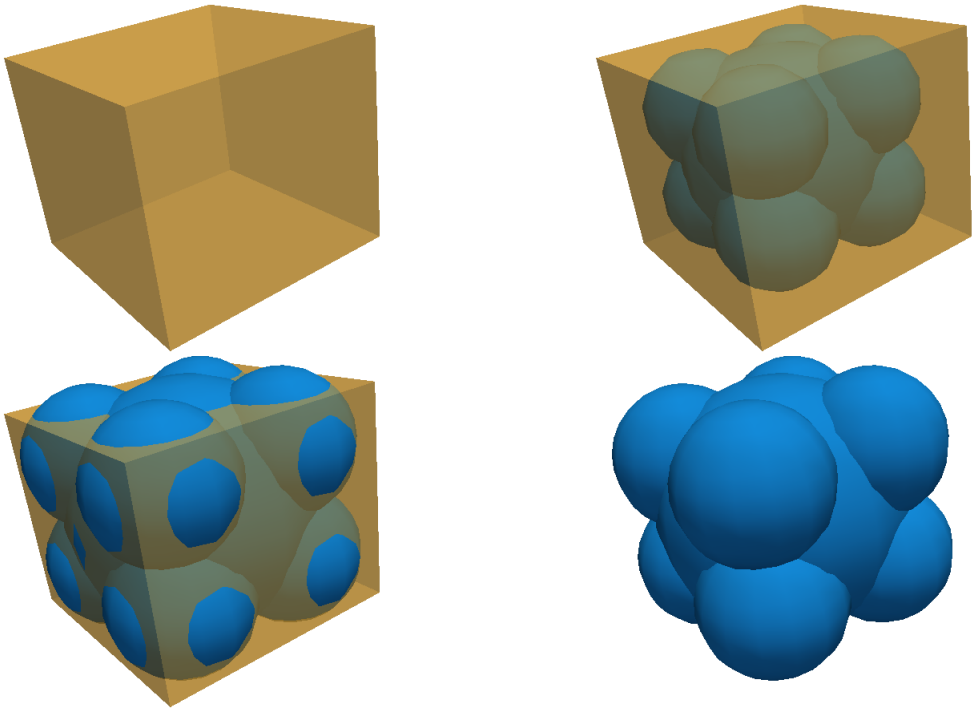


Figure 7.10: **Approximation of a box with 9 spheres**

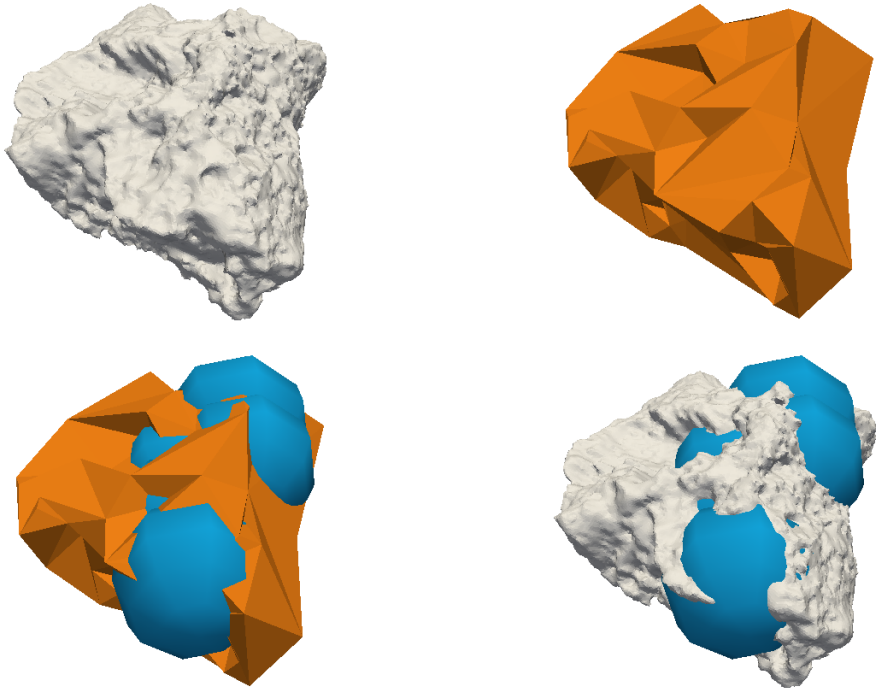


Figure 7.11: **Approximation of a NASA rock with 5 spheres**

7.2.2. Clusters of particles in HADES

In order to use a multisphere approach to handle fluid-particle simulations of non spherical particles it is necessary to implement in HADES the feature of rigid bodies motions: clusters of particles will move as rigid bodies. The idea is to sum all the forces that act on each particle of the cluster and integrate the motion according to this force.

To this purpose, two main features had to be developed: creation of clusters of spherical particles and a function that sums the forces and translates them with the creation of a torque. The structure of Modulus and Models of Jive creates a powerful environment, where these features can be developed exploiting the implementation of similar components of the softwares. In fact, in HADES a Model that creates ellipsoidal particles is given in `EllipsoidGenerator.cpp` and we will use it as basis to implement a generator for clusters of spherical particles, which we will develop in `ClusterGenerator.cpp`. Finally, considering `GravityModel.cpp` as a basis, we will develop a Model to translate all the forces that act on each particle of a cluster on the center of mass of the object and integrate its motion. This new model will be developed in `ClusterModel.cpp`.

The main idea is to split the detection of the contacts and the integration of the motion: contact detection will be performed by the spheres of the cluster, whereas the integration will be performed on a *ghost* particle placed in the center of mass of the object and with the inertial properties of the object. Afterwards, the real spherical particles will be translated and rotated accordingly to the outcome of the integration.

Generate Clusters : `ClusterGenerator`

The first step of the implementation of a multisphere method is to generate clusters of particles in the particle solver. To this purpose, we decide to extend the capabilities of the ellipsoid generator developed in HADES. The generator reads from the input file the specific details of the particles and it creates nodes, elements and bodies at the software level. For generating an ellipsoid, the generator takes the following info from the input file.

```
leftGrains = "ellipsoidGenerator"
{
     maxElemSize        =  20.0e-03;
     maxElemCurv        =     40.0;
     density            =   2200.0;

     groupName          = "leftGrains";
     outputFile         = "../DEM/files/leftGrains.gz";

     generation =
     {
       nrOfBatches      = 1;
       bodiesPerBatch   = 2;
       fireTime         = "0 + j * 0.00001";
     };
     shapeParams =
     {
       sRange             = [6.0e-01 , 6.0e-01];
       sdFunc             = "s";
     };
     displacement =
     {
       xFunc            =  0.0;
       yFunc            =  0.0;
       zFunc            =  "0.4 + i%2*1e-01";
       wFunc            =  0.0;
       axis             = [0.0, 0.0, 1.0];
     };
     velocity =
     {
       xFunc            =  0.0;
       yFunc            =  0.0;
       zFunc            =  "0.00 - i%2*1.5e-01";
       wFunc            =  0.0;
       axis             = [0.0, 0.0, 1.0];
     };
};
```

We decided to extend the generator and to generalize it for clusters of particles. The information on how many batches of bodies are inserted, how many bodies per batch are considered and the insertion time are not modified by the new structure. The same holds for the name of the group in which the particles will lie

and the initial conditions for position, velocity and angular position and angular velocity. The key point is that all these details will set the center of mass of the cluster.

With the purpose to create clusters of spheres, we added new features that can be read from the input file. The user can insert numeric values of 4 vectors (centerx, centery, centerz, radii). The first three vectors will contain the coordinates of each particle of the cluster with respect to its center of mass. The radii vector will, of course, contain the numerical values of the radii of the various sphere that form the cluster. The number of numerical values of these vectors has to coincide. In the input file, the user has to set also inertial informations. Two possibilities arise: to use the inertia of the object or the inertia of the multisphere cluster. In the case of arbitrarily shaped objects, both the evaluation will require a numerical integration pursued via Monte Carlo methods. The advise is to use the inertial information of the object. In this way, we are using the cluster approximation to detect contacts and the inertia of the object to integrate the motion. Note that, thanks to the volume correction adopted in the approximation algorithm, the mass will be conserved in the approximation. Hence, the translation motion will be independent to the choice of the inertia procedure. However, rotational motion will be strongly influenced by this choice. In HADES, the inertia information is given by a 4-component vector:

```
inertia = [volume   Ix/m   Iy/m   Iz/m];
```

where I_x, I_y and I_z are the component of the inertia tensor with respect to its principal inertia axis.

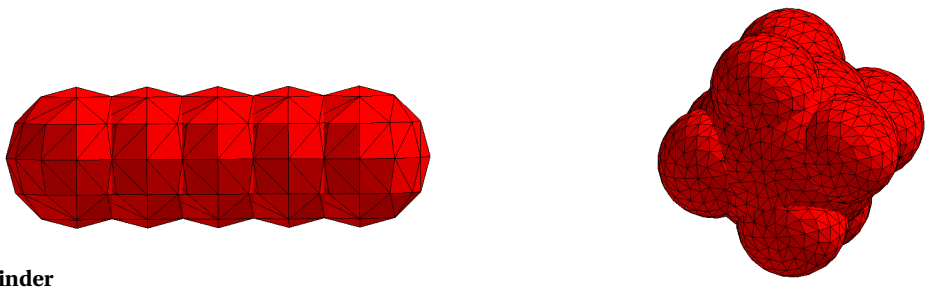
The new generator will create all the spheres of each cluster and it will assign them to the group specified in `groupName`. Each sphere will have the position defined by the position of the center of mass of the cluster which it belongs plus its relative position with respect to the center of mass. The velocity information will be the same of the center of mass and the inertia is evaluated as the inertia of a sphere.

Simultaneously, for each cluster, a *ghost* particle is created and assigned to the group specified in `ghostName`. These ghost particles are placed in the center of mass of each cluster and they are given the inertia of the object from the input file.

We now list the new information required by the Cluster Generator in the case of the creation of a cluster that approximates a cylinder.

```
ghostName     = "mask";
centerX       = [ -2.0e-02, -1.0e-02, 0.0, 1.0e-02, 2.0e-02];
centerY       = [ 0.0, 0.0 , 0.0, 0.0, 0.0];
centerZ       = [ 0.0, 0.0 , 0.0, 0.0, 0.0];
radii         = [ 0.01, 0.01, 0.01, 0.01, 0.01];
ghostInertia  = [ "3.14*1.0e-05"  ,                          // volume
                  "0.5*0.01*0.01" ,                          // Ix/m
                  "1/12*(3*0.01*0.01+0.1*0.1)",              // Iy/m
                  "1/12*(3*0.01*0.01+0.1*0.1)"  ];           // Iz/m
```

The outcome of the generation described by the code is provided in Figure 7.12(a). In Figure 7.12(b) we give the visualization of the approximation of a cube.



(a) **Cylinder**

(b) **Cube**

Figure 7.12: **Approximation in HADES via clusters of spheres**

Improve performance of `HertzContactModel`

The simplest idea would be to evaluate all forces acting on all the particles in the domain independently from their belonging to the same cluster or not, but using this approach leads to a waste of computational

resources. Hence, we slightly changed the implementation of the contact model to neglect all the contacts between particles that belong to the same cluster. In particular, we added two input fields to the model: `ghostNames` that receive as input the names of the groups made by ghost particles and `clusterNames`, which contains the name of the groups made by the respective clusters. Due to limited time in the implementation phase, some checks in the code are hard-coded for the case of just one type of clusters per simulation. This constraint should be eliminated without too much effort to provide a general framework for multisphere simulations. Despite the presence of this limitation, we managed to increase extensively the performance of the simulation, obtaining faster results with less evaluations, avoiding potential risks of numerical errors in useless calculations.

Integrate Clusters : `ClusterModel`

After the generation of the clusters and the ghost particles, a new Model is necessary for the evaluation of forces from the spheres in the clusters to the ghost particles and the transmission of the motion from the ghost particles to the spheres. This is pursued implementing two functions in the class `ClusterModel`.

The member function `ClusterModel::evalForces_` takes care of the translation of the forces from the spheres to the ghost particles. Let us consider a ghost particle inside the `ghostGroup`. The forces and the torques arising from contacts and acting on each spherical particle of the cluster are simply summed, but we have to be careful to the arise of torques when we translate each force from the spherical particle to the ghost particle, which is located in the center of mass. Let $\mathbf{r}$ and $\mathbf{F}$ be the *lever arm* vector and a force, respectively, then the torque $\mathbf{T}$ is given by:

$$\mathbf{T} = \mathbf{r} \times \mathbf{F}, \tag{7.12}$$

which can be written in matrix form as:

$$\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \det \begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ r_x & r_y & r_z \\ F_x & F_y & F_z \end{bmatrix} = \begin{bmatrix} r_y F_z - r_z F_y \\ r_z F_x - r_x F_z \\ r_x F_y - r_y F_x \end{bmatrix}.$$

Adding this arising torque to the summation leads to the desired results. In this way we obtained the correct forces acting on the ghost particles placed in the center of mass of each cluster. Now we eliminate every force acting on the spherical particles and we integrate the motion.

We highlight that the mapping between the different particles is pursued automatically thanks to the generation of the particles in the same generator and the successive division in different groups. For every batch of (possible multiple) bodies, spheres are generated first and the ghost particles afterwards and from the indexes of the particles it is possible to create a map between the two groups.

After the integration, the ghost particles are subjected to translational and rotational displacements. The necessary step is to move the cluster according to the outcome of the integration. In HADES, some general operations are organized in Actions, which can be performed by different Models. Models may or may not have their own implementation of these Actions to obtain *ad hoc* behaviors. An Action can be called by a Module and all the active Models that have that Action implemented will perform some operations. We exploit this framework using `VerletIntegrator` as module and `ClusterModel` as Model. To this purpose, after the integration, in the VerletIntagrator we call the Actions `UPDATE_VELOC_CLUSTER` and `MOVE_CLUSTER`. We implement these Actions only in the `ClusterModel`, therefore these Actions will be performed exclusively by this Model. In particular in `ClusterModel::takeAction` we implemented:

- `UPDATE_VELOC_CLUSTER`: we simply give to all the particles in a cluster the same velocity of the ghost particle, which is updated after the integration.
- `MOVE_CLUSTER`: we move the particles of each cluster accordingly to the outcome of the integration of the ghost particle. This is pursed in two steps: firstly, the particles are rotated with respect to the center of mass of the cluster which they belong and, secondly, they are translated.

Before continuing to describe the implementation of rigid body motions of the clusters in HADES, we give a brief introduction to the concept of quaternions, important mathematical tools which are used in HADES to perform efficient 3D rotations. Quaternions are mathematical entities proposed by sir W.R. Hamilton as an

extension of complex numbers. Formally, a quaternion can be expressed as:

$$\mathbf{q} = a + bi + cj + dk. \tag{7.13}$$

where a, b, c and d are real numbers and i, j, k are called *quaternion units* and they are the generalization of the imaginary unit i. The main properties of quaternions are that multiplication is non commutative and that $i^2 = j^2 = k^2 = ijk$. Mathematical details on their properties can be found in the original paper [23]. Properties in term of modern mathematics can be found in [29].

Quaternions provide a useful tool for many applications, from theoretical physics to mechanics, from computer graphics to robotics. The reason for their success is that they allow a robust representation of 3D rotations, avoiding the main problem that arise using Euler angles, i.e. *gimbal lock*. In fact, the usage of Euler angles provokes a rotation matrix with singularities, which translates, in practical applications, in the loss of a degree of freedom in the rotation of the system.
Particularly useful are the *unit quaternions*, i.e. $\|\mathbf{q}\| = 1$, which allow the straightforward computation of a 3D rotation matrix.

$$\mathbf{R} = \begin{bmatrix} 1-2(q_j^2+q_k^2) & 2(q_iq_j - q_kq_r) & 2(q_iq_k + q_jq_r) \\ 2(q_iq_j + q_kq_r) & 1-2(q_i^2+q_k^2) & 2(q_jq_k - q_iq_r) \\ 2(q_iq_k - q_jq_r) & 2(q_jq_k + q_iq_r) & 1-2(q_i^2+q_j^2) \end{bmatrix}. \tag{7.14}$$

The rotational matrix can be used to rotate a 3-component vector $\mathbf{v}$ through $\hat{\mathbf{v}} = \mathbf{Rv}$. This feature is exploited extensively in HADES to obtain efficient rotations of particles as results of their motions.

Let us now go back to the HADES level to describe the rotational displacement of clusters of particles. To this purpose, initially each particle is given:

- position relatively to its center of mass (from input file)
- angular inclination of the ghost particle after the motion

Afterwards, the rotational matrix (7.14) is calculated considering the quaternions that describe the angular inclination of the ghost particle. The position of each particle with respect to the center of mass of the cluster which it belong is evaluated performing the matrix-vector multiplication $\hat{\mathbf{v}} = \mathbf{Rv}$. The result is the new position of the sphere with respect to the center of mass of the cluster. The rotational displacement has been concluded. As far as the translational displacement is concerned, it is just necessary to sum the outcome of the rotation of each sphere to the position of the ghost particle.

With these operations we achieved the motion of clusters of spheres as rigid bodies in HADES. In the following Chapter we test the implementation of the multisphere approach with two numerical experiments: pure DEM and coupling of CFD and DEM for cubes, approximated with clusters of spheres adopting the approach that we described in this Chapter.

III

Tests and Applications

8

Test of Implementations

In the previous Chapters new features have been developed to achieve improvements on the available resources for fluid-particles simulations: the multisphere approach has been developed and implemented in HADES to handle non spherical particles, whereas the coupling between OpenFOAM and HADES has been obtained adapting an existent framework (CFDEM). The step forward on the project is to perform some test cases, comparing the numerical results achieved by the new software tools with experiment with the same (or similar) setting available in literature.

Several cases are available in literature to compare the accuracy and the performance of the multisphere implementation. As far as the interactions between clusters of spheres are concerned, studies have been carried out on the measurements of filling and discharging of particles in boxes or hoppers: falling particles give the opportunity to test collision and packing properties. As far as fluid-particle interactions are concerned, flow around objects has been studied and accurate and reproducible results are available for simple cases.

We propose two test cases, one focused on the multisphere approach to approximate the discharge of a box filled with cubes and one focused on the fluid-particle interaction, in particular on the differences that arise on the flow if a square cylinder or its approximation with a cluster of spherical cylinders is considered.

8.1. Discharging of Cubes

In this Section we test the implementation of the multisphere approach in HADES comparing the results of the simulation with numerical results available in the literature. In particular, we decide to reproduce the numerical experiment that was performed in [15], where the authors provide results of validation of their simulation with experimental results. The filling and the successive discharge of a hopper is studied.

In order to have comparable results, the setting of the experiments have to be equal, or at least very similar. We now describe the geometry of the discharger and give the physical properties of the setting chosen by the authors of the paper. The hopper is rectangular-shaped and its dimensions are given in Table 8.1. In Figure 8.1 the hopper used in the simulation is showed. Particles are inserted at the top of the hopper, through an empty face. In the Figure, the bottom in the phase of discharge is given. The bottom is initially closed in the phase of filling, as it can be seen in Figure 8.2(a), whereas during the phase of discharging an orifice is considered (see Figure 8.2(c)). The size of orifice is again given in Table 8.1. Due to a lack in the implementation of contacts between spheres and *finite* planes, it is necessary to approximate the finite planes with beds of spheres. The planes involved are shown in 8.2(b) and 8.2(d), for the closed and open box, respectively. We will discuss the effects of this approximation in the following paragraphs.

The aim of this test is to compare the discharging time, the discharge flow and the related quantities to the results presented in the paper. The particles analyzed are spheres and cubes. We will approximate the cubes with clusters of spheres in HADES. Physical properties of particles, as well as dimensions for spheres and cubes are given in Table 8.2. In particular, we are considering particles with the same value for the edge of the cube and the diameter of the spheres. Note that this means that the volume (and therefore the mass)

of spheres and cubes are not the same. The particles are made of acrylic, whose physical and mechanical properties can be found in appropriate online databases, such as http://www.matweb.com [1].

In order to avoid dependence of the results from the initial configuration, we insert the particles in the hopper at the same height, but with a random orientation and we perturb the x and z component of the initial position with a sampling from a uniform variable.

Property	**Value**
Hopper Height	500 mm
Hopper Width	200 mm
Hopper Thickness	30 mm
Orifice Width	60, 80, 116 mm
Orifice Thickness	30 mm

Table 8.1: **Geometric properties of hopper**.

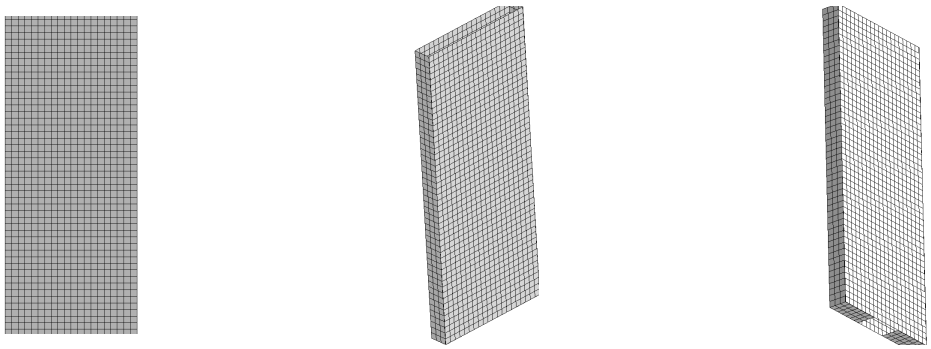
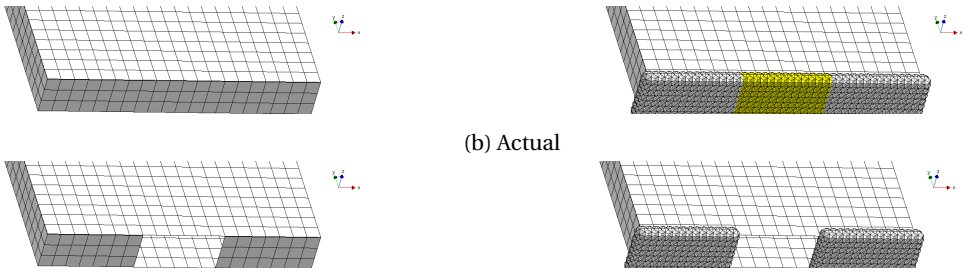


Figure 8.1: **Geometry of Hopper**.



(a) Ideal (b) Actual

(c) Ideal (d) Actual

Figure 8.2: **Bottom Domain Details**. Approximations with bed of particles are necessary due to a lack in the implementation between particles and finite planes.

A priori observations

Before running the simulations and analyzing the results, it is necessary to highlight some aspects that can influence the accuracy of the results and lead to discrepancies between the results of the current and the ones presented in literature.

First of all, in the paper taken as reference, the authors apply a Hooke's model for the elastic contacts, with a fixed stiffness. Moreover, they use a damping force only in the normal direction of the contact. Instead, in our simulation we are using a Hertzian Model, with non-constant elastic and damping contributes both in

[1] In particular, general purpose acrylic data can be found at http://www.matweb.com/search/DataSheet.aspx?MatGUID=3cb08da2a0054447a3790015b7214d07

Property	Value
Spherical Diameter	12 mm
Cube length	12 mm
Particle density	$\rho = 1200\,\mathrm{kg\,m^{-3}}$
Normal damping Coeff	$\eta_n = 3000\,\mathrm{m^{-1}\,s^{-1}}$
Tangential damping Coeff	$\eta_t = 3000\,\mathrm{m^{-1}\,s^{-1}}$
Friction Coeff	$\mu = 0.4$

Table 8.2: **Particle Properties**.

the normal and in the tangential direction. In HADES, the damping is modeled as proportional to the normal and tangential displacements and the multiplying factor has to be inserted by the user. Unfortunately, as discussed in Section 2.1, as far as the damping coefficient is concerned, there is a lack of accurate models in literature and it is usually used as tuning parameter. For an accurate tuning of this parameter experimental data would be required, at least to validate the order of magnitude of the coefficient. A valid approach would be to validate the coefficient for spheres of comparable dimension of arbitrarily shaped objects with the same mechanical properties, but this type of validation was not possible in the limited time framework of this project. In the present simulation, we tune the order of magnitude of the damping coefficient avoiding nonphysical oscillations in the simulation, but we have to keep in mind that this parameter not accurately tuned will introduce a consistent error in our simulation.

Secondly, the current implementation of HADES allows contact evaluations between spheres (or clusters) and *infinite* planes. This is enough in the case of filling of boxes, whereas if we want to study a discharging due to the presence of an orifice, the implementation is not appropriate. To overcome this problem, we consider an approximation of the bottom plate with a bed of spherical particles which are constrained to maintain their position. In this way we can simulate the filling phase. Afterwards, we will fix a set of particles of the bottom bed to move to create an opening of the same size of the orifice. This approximation allows us to simulate the discharging phase capturing the edge effects.

Finally, we have to consider that the results shown in the paper are obtained with a numerical implementation of DEM specific for the cubic case. The cube is an extremely stable geometrical shape and this property has strong effects in the packing of particles. Approximating cubes with clusters of spheres we introduce strong instabilities in the packing phase. We expect the packing to be more structured than the packing of spheres, but expecting a very accurate results in the case of cubes is unfortunately not realistic. Packing experiments with more unstable geometrical shapes would be more fair for the validation of the multisphere approach, e.g. considering ellipsoids or corn-shaped particles. We selected the cube experiment to analyze the behaviour of the approach in a very extreme case.

Spheres

We firstly consider spherical particles. The filling phase gives a stable configuration, which can be seen in Figure 8.3: the steady state solution for the filling phase is reached after $t = 1$s of simulation time. Afterwards, at time $t = 2$s, we open an orifice in the bottom plate and we study the flow-rate of the discharging of the hopper. In Figure 8.4 we plot the percentage of mass discharged versus simulation time, for different sizes of the orifice. The results of the current simulation are reported in dotted lines, whereas the results obtained in the reference paper are shown in straight lines.

Qualitatively, we observe that the discharge is faster than the results from the ones given in [15], but they follow a very similar evolution in time. In particular, the distance between the results is larger with the smaller size of the orifice, whereas we get very close results when the orifice reaches its maximum value.

Possible explanations for the deviations of the results are: influence of the initial configurations, which are for sure slightly different between the two cases, and the usage of different contact models used, as describe in the previous paragraph. Due to the different model adopted, precise quantitative results are hard to obtain since an accurate tuning of the parameters would require extensive experimental data. Nevertheless, we can

claim that the macroscopic behaviour is compatible between the two different models at the resolution that we accept.

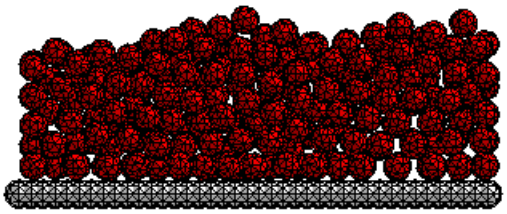


Figure 8.3: **Filling of Spheres**.

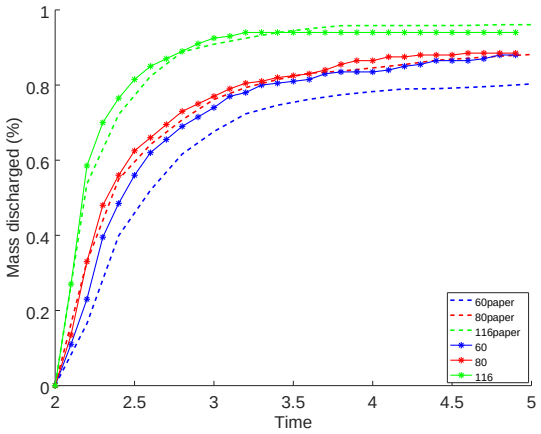


Figure 8.4: **Discharging of Spheres**. The different colors describe differente orifice openings, as stated in the legend. The dotted line shows results from the paper, the straight line describes data of the current simulation.

Cubes

We now analyze the performances of the approximations of cubes with clusters of spheres. In order to obtain the cluster, we run the algorithm that we implemented in the Chapter on Non-Sphericity 7 using an STL file with a cube of edge 12 mm. We obtain a cluster with the following radius and positions of centers of spheres (with respect to its center of mass). The cluster is shown in the left part of Figure 8.8.

```
centerX = [ 0.000,  0.0032, -0.0032, -0.0032,  0.0032, 0.0032, -0.0032, -0.0032,  0.0032];
centerY = [ 0.000,  0.0032, -0.0032,  0.0032, -0.0032, 0.0032, -0.0032,  0.0032, -0.0032];
centerZ = [ 0.000, -0.0032, -0.0032, -0.0032, -0.0032, 0.0032,  0.0032,  0.0032,  0.0032];
radii   = [ 0.0065, 0.00365, 0.00365, 0.00365, 0.00365, 0.00365, 0.00365, 0.00365, 0.00365];
```

We run the simulation and we reach an equilibrium at $t = 1.1s$. The equilibrium is shown in Figure 8.5. We observe that the packing of the simulation is unfortunately different than the results presented in the reference paper. In fact, in the paper the authors obtain a very structured packing of cubes, whereas in our simulation the packing resembles the shape of a V. The height of the packing is similar, but the equilibrium with the shape of a V is much more unstable when we consider the discharging of the box. This translates in a more consistent discharging when the orifice is opened. In our simulations two situations occur:

- Arches formation (see Figure 8.7)
- Discharging of 70 - 90 % of particles, depending on the opening of orifice

In the paper, basically two situations occur:

- Arches formation
- Discharging of (approximately) only the particles that lie in the zone above the orifice

The authors obtain these different two behaviours and they propose as results the data averaged between these cases (in particular 1 arch formation and 2 pure discharges). The results form the paper are shown in Figure 8.6 with the dotted line. In the same figure we plot with a straight line the discharging ratio in one case in which arches did not occur.

It has to be noticed that the randomness introduced in the initial position and orientation produces configurations in which arches do or do not show up. Consequently, the discharged mass can vary a lot, but the authors of the paper claim that the maximum of the discharged mass is close to the percentage of mass above the orifice. The results of the current simulations are in agreement with the literature data up to time

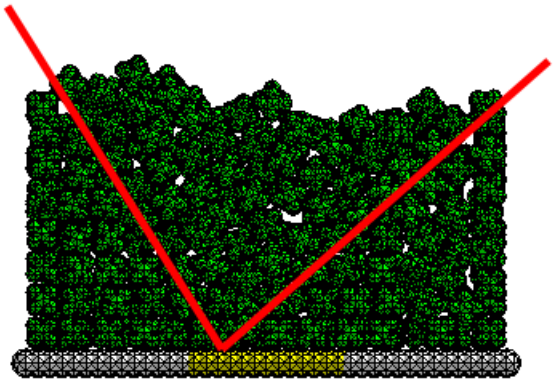


Figure 8.5: **Filling of Cubes**.

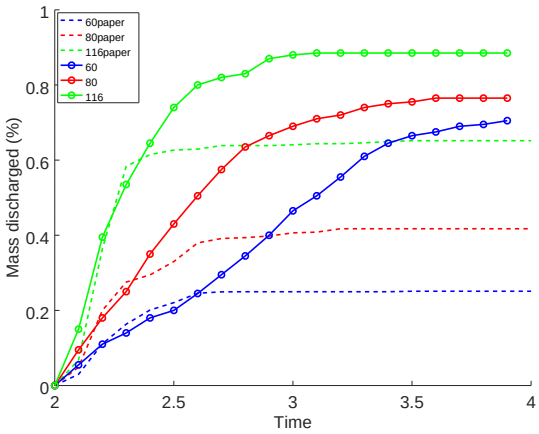


Figure 8.6: **Discharging of Cubes**. The different colors describe differente orifice openings, as stated in the legend. The dotted line shows results from the paper, the straight line describes data of the current simulation.

$t = 2.3/2.4$, meaning that we manage to capture reasonably well the first phase of the discharging, which intuitively does not depend on the packed configuration. When the packed configuration starts to exert its influence, results deviate giving a strong discharging rate in the simulation, if compared to the data from the paper.

The deviation in the packing phase can be influenced by a variety of phenomena. Firstly, the approximation of cubes, *extreme* stable geometries, with spheres will introduce instabilities on the packing. Secondly, the different models implemented in the simulation cause deviation of results. Moreover, we remind that the damping coefficients are modeled in literature in very different ways and in the present simulation we estimate only their order of magnitude. Additionally, one could think that the approximation of the bottom plate with spheres could play a role, but after having performed packing test with pure planes (possible, as stated before, only for the filling phase where there are no orifices in the surface of the box), we claim that the approximation of the bottom plate influences strongly the dynamics but it does not influence the packing properties: the equilibrium configuration obtained with a regular plane at the bottom presented a similar V structure as the particle-bed case. Finally, an additional factor that could influence the packing is the configuration of spheres inside the cluster. Let us consider a cube approximated with 9 spheres. If we consider a face, there can be 3 different configurations as outputs of the algorithm approximations: 1, 4, 5 spheres sticking out from a face, see Figure 8.8. This definitely plays an relevant role in the contact dynamics and in the packing configuration.

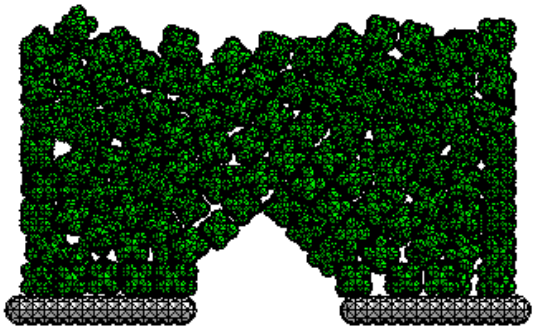


Figure 8.7: **Arch Formation with orifice of 60** mm.

Comments

Analyzing the outcome of the numerical simulation, we emphasized some discrepancies between the results presented in the reference paper and our numerical simulation. Several hypothetical reasons that could explain the differences were described. It has to be highlighted that the situation that we wanted to verify was extreme in the spectrum of the possible configurations, since the cube presents extreme properties of stability under the presence of forces and probably this situation is the case in which the multisphere DEM approach shows its worst performances. We have to remind that the multisphere approach can be used with an arbitrarily shaped object, therefore its best feature show up in the irregular cases, where no other methods can be implemented. Due to the limited amount of time of the project we could not proceed to further investigations, but a strong suggestion for future researches is to use the multisphere approach for more unstable geometrical shapes and study the packing of ellipsoids or rounded-shape objects to pursue the validation of the results. Literature is available for such configuration, for example [49] for the packing of ellipsoids, whereas [44] deals with corn particles. The results for the contacts, the initial discharging and the formation of arches seem promising to achieve the validation of the implementation.

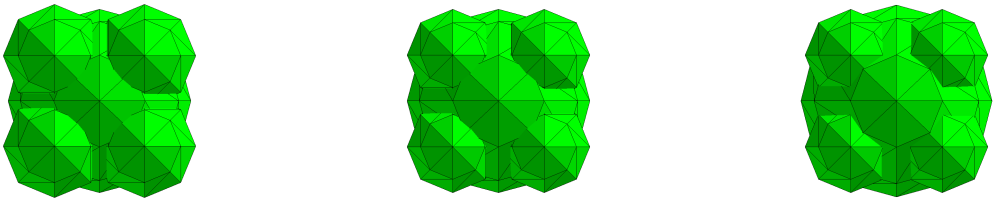


Figure 8.8: **Possible Clusters** Filling dynamics will depend on how many spheres compose the outer layer of the cluster: 4, 5 or 1 respectively in the figure above

8.2. Flow around a square cylinder

In this Section, we set up a numerical experiment to study the effects on the flow patterns of the approximation of cubes with a cluster of spheres. More precisely, we analyze the flow around a square cylinder, a circular cylinder and an approximation of a square cylinder with circular ones. For all the cases we compare flow patterns and the drag coefficient for a range of Reynolds numbers and we compare the results with literature, in particular paper [7]. To this purpose, we set up the numerical experiment reproducing the setting of the results presented in the paper. We consider a rectangular 2D domain and a cylinder inside it. Note that since OpenFOAM allows only 3D computations, we give to the domain a thickness, but the simulation will still be 2D using appropriate boundary conditions (empty faces) on the third dimensional direction.

Domain

The domain is a 2D rectangular box, with length L and height H. A square cylinder, of edge D, is put at the half of the channel height and at a quarter of the channel length. The blockage ratio is evaluated as $B = D/H$. The setting is summed up in table 8.3. In particular, we choose some values for the dimension that satisfy these constraints. The values that are used in the calculation are shown in table 8.4. In Figure 8.9 we show the mesh used for the calculations. Since around the cube some instabilities may occur, we choose to refine the mesh around it, for a distance equal to the half of its size: this will give smoother solutions and prevent numerical instabilities. The mesh around the cube is presented in Figure 8.10.

Boundary Conditions

In order to achieve a solution to the problem, which is described by Navier-Stokes equations, we need to set appropriate initial and boundary conditions. In particular, we consider a fluid at rest (zero initial velocity field and zero pressure field) as initial condition. A velocity is prescribed at the left of the domain in the x direction, therefore the fluid will flow from the left to the right. We now describe in detail the boundary conditions that we impose.

Property	Value
Channel Length	L
Channel Height	H
Square Diameter	D
Blockage ratio	$B = D/H = 1/8$
Inflow Length	$l = L/4$

Table 8.3: **Geometric properties of fluid domain in the reference paper**.

Property	Value
Box Width	$L = 200$ mm
Box Height	$H = 500$ mm
Box Thickness	$T = 30$ mm
Square Width	$D = 60$ mm
Square Thickness	$T = 30$ mm

Table 8.4: **Geometric properties of fluid domain in the calculations**.

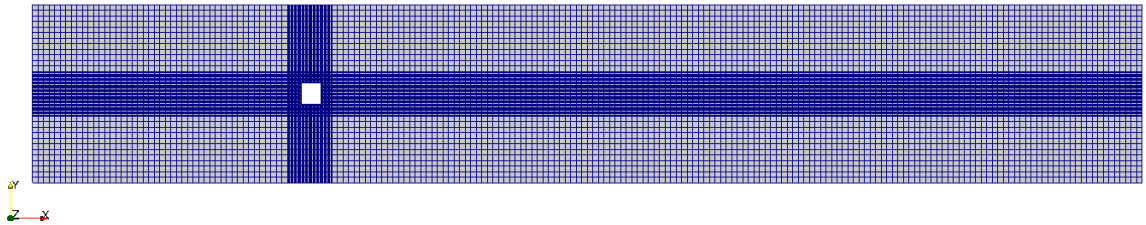


Figure 8.9: **Domain**.

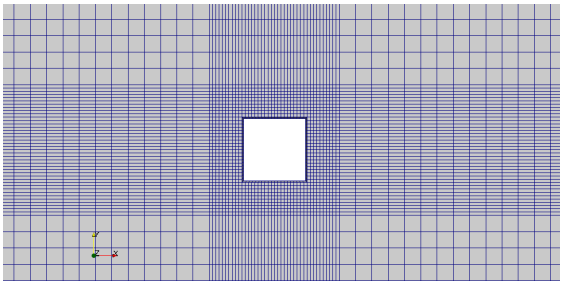


Figure 8.10: **Domain Detail**.

Wall

The top and the bottom of the domain shown in Figure 8.9 are considered as walls. In literature, this case is named *no slip* boundary condition and it acts imposing the velocity of the fluid to 0 and the fix the gradient of the pressure to 0.

Inlet

As inflow, we consider a parabolic profile for the velocity. At the two extremes (intersections with the walls) the velocity is forced to be 0, whereas at the middle of the patch it reaches its maximum value u_{max}. The velocity is therefore described by the following expression:

$$\mathbf{u}(x,y,z) = \begin{bmatrix} u(x,y,z) \\ v(x,y,z) \\ w(x,y,z) \end{bmatrix} = \begin{bmatrix} -\dfrac{u_{max}}{0.04^2}y^2 + \dfrac{2u_{max}}{0.04}y \\ 0 \\ 0 \end{bmatrix} \tag{8.1}$$

The velocity profile in the case of ($u_{max} = 0.03\,\mathrm{m\,s^{-1}}$) is shown is Figure 8.11. Again, a homogeneous Neumann boundary condition is imposed to the pressure.

Outlet

Since, from experimental and theoretical results available in literature, we expect the formation of vortexes in the flow, we impose an advective boundary condition at the outflow. This assures that the vortexes will flow and reflection in the inner part of the domain is avoided. The general advective condition reads:

$$\frac{\partial u_i}{\partial t} + u_{adv}\frac{\partial u_i}{\partial x} = 0, \tag{8.2}$$

and in particular we choose the advective velocity u_{adv} to be equal to the maximum velocity u_{max} imposed at the inlet. As far as pressure is concerned, a homogeneous Dirichlet condition ($p = 0$) is chosen.

Empty

The mathematical formulation of a 2D problem does not require a boundary condition in the third dimension, but in OpenFOAM a boundary condition is required since all the simulations are 3D. Hence, an empty boundary condition is imposed to all the patches that span the third dimension. With this boundary condition, the solver does not perform any calculation in the direction of the normal of the empty faces: the simulation becomes 2D.

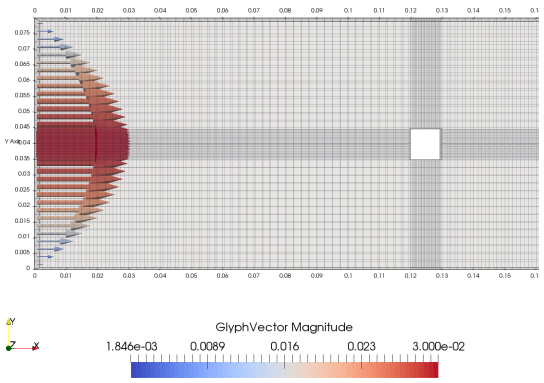


Figure 8.11: **Parabolic Inflow at** $u_{max} = 0.03\,\mathrm{m\,s^{-1}} \Rightarrow Re = 300$.

Fluid parameters and dimensionless numbers

In Fluid Dynamics, very often flows are analyzed in dependence of dimensionless numbers, which manage to describe the behavior and the pattern of flows in various situations. In particular, one of the most relevant dimensionless quantities is the *Reynolds number*, which is defined as the ratio of inertial and viscous force

that act on a fluid. Let ρ be the density of the fluid, μ its dynamic viscosity, U a characteristic velocity and d a characteristic length, then the Reynolds number is defined as:

$$Re = \frac{\rho U d}{\mu} = \frac{U d}{\nu}, \tag{8.3}$$

where ν is the kinematic viscosity and it is defined as $\nu = \mu/\rho$. We choose as reference length the size of the square cylinder and as characteristic velocity the maximum of the parabolic profile at the inlet u_{max}. Hence, the Reynolds number will be evaluated as:

$$Re = \frac{u_{max} D}{\nu}. \tag{8.4}$$

In order to perform simulations with a variety of the Reynolds Number, we decide to keep the physical properties of the fluid and the dimension of the domain as constants, whereas we vary the maximum of the inlet velocity. In Table 8.5 we give the values of the properties of the fluid that we considered in the simulation. The density and the kinematic viscosity are very close to the properties of water. The only parameter that we can modify to change the Reynolds number is therefore the maximum of the inlet parabolic profile. In Table 8.6 we give some reference value for the relation between the two parameters for the most important cases that we will analyze in the numerical results.

Property	**Value**
density	$\rho = 10^3$ kgm^{-3}
dynamic viscosity	$\mu = 10^{-3}$ kgs^{-1}m^{-1}
kinematic viscosity	$\nu = 10^{-6}$ m^{2}s^{-1}

Table 8.5: **Fluid Properties**.

Max Velocity [ms^{-1}]	**Reynolds**
0.0001	1
0.003	30
0.01	100
0.03	300

Table 8.6: **Relation between u_{max} and Reynolds Number**.

The physical system under consideration is an object (square or circular cylinder) inserted in a fluid. The application of the conservation of momentum to the system leads to the rise of forces that are exerted by the fluid on the object. The most common strategy to evaluate these forces is to divide them in components which are parallel or perpendicular to the relative velocity of the fluid and the solid body: we define the first as *Drag force* and the second as *Lift force*. These forces can be caused by three different physical processes: viscosity, a gradient of pressure and porosity of the object that is hit by the fluid. In the present application, we neglect the porosity of the material, hence we are interested on the drag force that arise from the first two phenomena. Their influence on flow patterns and arising forces will be strongly dependent on the Reynolds number.

In particular, we define a dimensionless number that is proportional to the drag force, in order to simplify the comparison between different flow settings. Let D be the drag force acting on the object, u a reference velocity of the flow, A a reference area (an estimation of the contact area between the fluid and the solid body in the direction that we consider) and ρ the density of the fluid. Then the (dimensionless) *drag coefficient* is defined as:

$$C_d = \frac{2D}{\rho u^2 A} \tag{8.5}$$

In the presence of viscosity, if we perform a dimensional analysis of the system we obtain that the drag coefficient depends on the Reynolds number Re. Their dependence will be studied for the square cylinder, the circular cylinder and the approximation of square with circles. The numerical results will be compared with experimental or numerical results available in literature.

Literature Flow around a Circular cylinder

The flow around a circular cylinder was deeply studied and analyzed in the XX century, both theoretically and experimentally. In modern textbooks of Fluid Dynamics the case is often presented as an example of complex flow patterns that can arise from very simple geometries. We briefly sum up the experimental data available in three of the most common and exhaustive monographs on the subjects: [4], [28] and [35].

The flow around a circular cylinder surrounded by a viscous fluid is strongly dependent on the Reynolds number: it causes dramatic changes in flow patterns and physical quantities. We distinguish 5 intervals of the Reynolds number in which the behavior assumes very different connotations:

- $Re < 4$. This case correspond to the *Stokes flow*, or *creeping flow*. The flow is attached, the inertia effects are very small and there is a balance of the viscous and pressure terms. This equilibrium causes the vorticity created by the no slip condition at the wall of the cylinder to be diffused and not advected.
- $4 < Re < 40$. The flow is not attached anymore and two stable vortexes, also called *eddies* arise at the back of the cylinder. Nevertheless, the streamlines remain (almost) symmetrical and the flow is considered stable.
- $40 < Re < 80$. The vortexes that arise behind the circular cylinder are not stable anymore: periodic oscillations of the velocity in time appear. This results in the formation of oscillation vortexes known as *von Karman vortexes*, that appear in constant positions. The eddies behind the cylinder and the *von Karman* vortexes do not interact.
- $80 < Re < 3 \cdot 10^5$. The flow keeps the same configuration except that the eddies interact between each other: the eddies develop closer to the surface and the *von Karman* vortexes start to oscillate. Increasing Re yields the transition of the *von Karman* vortexes to turbulent wakes, but the boundary layer is maintained laminar and stable.
- $3 \cdot 10^5 < Re < 3 \cdot 10^6$. The boundary layer starts to be unstable.
- $Re > 3 \cdot 10^6$. The boundary layer is completely turbulent.

For all this cases, complete data are available for the values of drag coefficient. We will plot the values for the relevant cases below when we will compare the numerical results with literature data.

Literature Flow around a Square cylinder

For the case of a square cylinder, less data are available in literature. We will consider the data presented in [7], the paper that we used as reference for the setting of the test case, for Reynolds numbers in the range of $[0.5 : 300]$ and [30] for data at $Re = 500, 1000$.

- $Re < 5$. The flow is in the regime of the *creeping* flow, we have the same dynamics as the circular case.
- $5 < Re < 60$. Eddies start to develop behind the cylinder and the flow separation starts to happen. The flow is stable
- $60 < Re < 100$. The *von Karman* vortexes start to appear: oscillations rise and the flow is unstable. In this range, the eddies and the *von Karman* vortexes do not interact
- $100 < Re < 1000$. Interaction between eddies and *von Karman* vortexes: the separation happens in the leading edge of the cylinder.
- $Re > 1000$. No data was found.

It is straightforward to notice that the dynamics of the flow on square and circular cylinders is very similar, except for the small delay on the square case: considering the increasing of Re, the round shape of the circle seems to anticipate the development of the flow, if compared to the sharp shape of the square. Data for the drag coefficient in dependence of the Reynolds number is available for the interval considered in the present simulation, i.e. $Re \in [1, 500]$, and will be plotted with the numerical results, to compare the accuracy of the simulation.

Numerical Results and comments

We now compare the numerical result that we obtain from the simulation to the the experimental and theoretical results available in literature. We choose to perform the simulation at 6 different Reynolds Numbers: $Re = 1, 30, 100, 200, 300, 500$. We analyze the macroscopic behaviour of the fluid, highlighting the patterns of the flow and we evaluate the drag coefficients and we plot them against the data found in literature.

In the current simulation, we compare the following 3 situations:

- Flow around a square cylinder through a holed-mesh, as stated in Figures 8.9 and 8.10.
- Flow around a circular cylinder through fluid-particle coupling: we place a sphere in the middle of the domain. Since the simulation is 2D we will obtain a circular cylinder with the correct radius inside the mesh.
- Flow around a cluster of spherical cylinders, inserting a clusters of spheres at the correct height in order to have the accurate approximation of a cluster of cylinders.

In order to perform these simulations, we use different solvers in OpenFOAM.

- Holed Mesh
  We use the standard `pimpleFoam` implemented in OpenFOAM. In this case, no interactions with particles are considered. The solver allows to have a non fixed CFD timestep ΔT_{CFD}, using properties of the solvers `simpleFoam` and `pisoFoam`. A constraint on a maximum Courant Number (or CFL) is imposed.
- Circular Cylinder
  We use the CFDEM solver `cfdemSolverIB` which is based on the solver `pisoFoam`. The implementation of the solver does not allow to have dynamic time steps for the CFD evaluations, therefore it is important to keep manually the CFL number less than 1. That is why we fix the ΔT_{CFD} as constant through the entire simulation. This imposes a strong constraint on the accuracy of the simulation. In fact, initially we require a small time step to obtain an appropriate reaction of the fluid which at time $t = 0$ is at rest. A larger time step could be decided for the following part of the simulation, with the exception of the cases when vortexes show up. In these cases a fine time step is again necessary.
- Cluster of Circular Cylinders
  The solver used is the same of the circular cylinder case, where constraints become even worse since we are approximating more cylinders in the CFD grid and errors of shape approximation become more relevant.

In Table 8.7 we show the time steps and the maximumu CFL number reached for each simulation and each solver used.

Case	**Holed**		**Circular**		**Cluster Circular**	
Solver	`pimpleFoam`		`pisoFoam CFDEM`		`pisoFoam CFDEM`	
	ΔT_{CFD}	CFL_{max}	ΔT_{CFD}	CFL_{max}	ΔT_{CFD}	CFL_{max}
Re 1	–	1	0.001	0.0001	–	–
Re 30	–	1	0.001	0.009	0.001	0.0096
Re 100	–	1	0.001	0.034	0.001	0.0342
Re 200	–	1	0.001	0.073	0.001	0.0754
Re 300	–	1	0.0005	0.0552	0.0005	0.0580
Re 500	–	1	0.0005	0.092	0.0005	0.0913

Table 8.7: **Time steps and CFL numbers for solvers used**.

We now describe the outcome of the current simulation. Firstly, it is necessary to stress that unfortunately the solver `cfdemSolverIB` is not completely reliable at low Reynolds numbers. The developers of the solver, state in [17] that the Immersed Boundary Method implemented is not accurate when Reynolds number approaches 1, whereas its results become reliable when we consider moderate Re. To overcome this issue, a

new solver, `cfdemSolverForceIB` or pisoIB, has been developed in [6], but it is not yet available in the open-source version of the software. A further implementation of this new method would be essential in cases characterized by a low Reynolds number.

Hence, we expect some differences between the theoretical and numerical results when $Re = 1$ when we are considering the circular cylinder and the cluster of cylinders with the fluid-particle framework.

Firstly, let us consider the flow patterns, analyzing streamlines of the flow at different Re in the cases of square, circular cylinders and cluster of circular cylinders to approximate a square. We have to keep in mind that comparing flow patterns of different geometrical shapes and in particular with approximated ones (the circular cylinder and the cluster are interpolated in the CFD mesh via the void fraction scalar field) is not an accurate measure of accuracy of the solver, but at least we can observe the flow behavior and compare qualitatively the results with expectations from theory. In Figure 8.14, 8.15 and 8.16 we plot the streamlines of the flow for different Re. We remind that streamlines are defined as lines that are tangential, in every point, to the velocity field. The flow patterns of square and circular cylinder agree very well with the patterns described in the previous paragraph, with the correct macroscopic behaviors. We observe that also the approximation of the square cylinder with the cluster of circular cylinders shows a reasonable behavior.

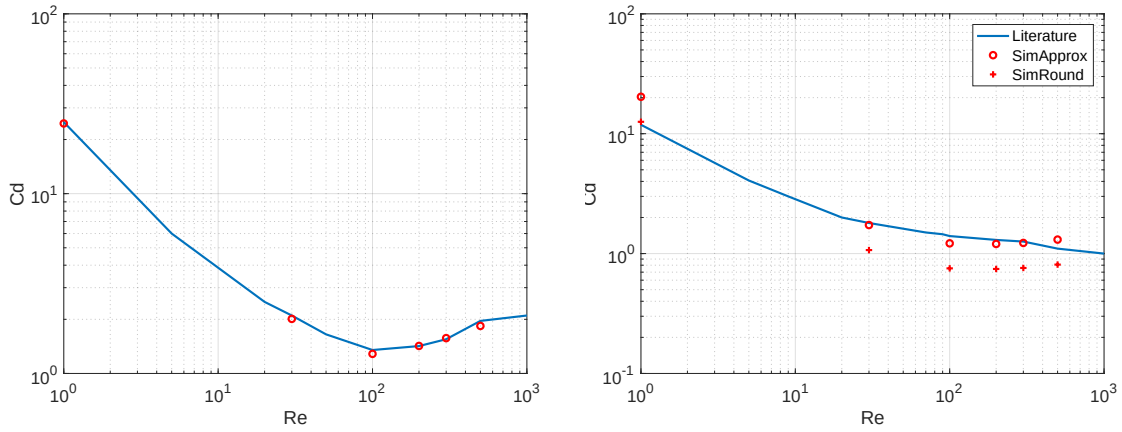
In order to analyze the accuracy of a solver, a measure of the drag coefficient is often chosen in literature. Hence, we evaluate the drag coefficient, as defined in the previous paragraph.

We firstly consider the case of the square cylinder obtained by the holed mesh and we evaluate the drag coefficient of the simulation varying Re. Data are extracted by `functionObjects` in OpenFOAM. In Figure 8.12(a) we plot the drag coefficient in dependence of the Reynolds number. A very good agreement with data from literature is observed.

Secondly, we compare the behavior of the drag coefficient in the case of a circular cylinder and we compare the result obtained with the framework that we developed with data available in literature for flows around circular cylinders. The comparison is performed in Figure 8.12(b). First of all, we have to notice that the drag coefficients depends on a reference area A, which has to be chosen as an relevant parameter to the problem. Since we are using the fluid-particle simulation framework, the perfectly rounded surface of the circular cylinder is approximated by the void-fraction field in the CFD mesh. This will cause discrepancies in the reference area considered. In the Figure, we plot both the result evaluated with the rounded reference area and the one obtained with the approximation through interpolation on the CFD grid. We observe that if we take as reference the approximated area we have results that are closer to the data from literature. It is straightforward to notice that a large deviation from the literature data is present in $Re = 1$, but this is explained by the lack of accuracy of the solver in the case of low Reynolds numbers. The accuracy increases considering moderate Reynolds numbers, in particular in the cases $Re = 30, 100, 200$. In the last two cases ($Re = 300, 500$), we see that the drag coefficient does not decrease as expected, but stays stable. An explanation for this behavior may be that the CFD time step is larger than the adequate one and that accuracy of the solution is therefore not adequate. A good strategy for investigation of accuracy may be to vary the time step keeping the CFL number comparable in all the cases under consideration. Globally, we can be satisfied by the behaviour of the solver at moderate Reynolds number.

Finally, we analyze the case of a cluster of circular cylinders that approximate a square cylinder and we study the behavior of the drag coefficient in Figure 8.13. Due to the lack of accuracy at low Re, we focus on the interval $30 \leq Re \leq 500$. We see that in this interval the drag coefficient evaluated using the approximated reference area manages to follow the behavior of the theoretical drag coefficient for square cylinders, whereas we overestimate the coefficient if we use the rounded reference area. The small discrepancies are caused mainly by the approximation of the rounded shape in the CFD cells through the void-fraction scalar field. We also observe that this approximations tend to emphasize numerical instabilities, therefore the presence of vortexes and eddies is anticipated if compared with the perfect square cylinder case.

Since our aim is to perform simulation with arbitrarily shaped particle, we can be satisfied by the degree of accuracy reached for a regular object. We in fact expect that the accuracy of the approximation increases with the non regularity of the object that we consider: if an object does not have symmetrical or regular properties, the instabilities caused by the approximation with spheres into the void-fraction field will be less relevant, since they will be intrinsic properties of the system. We can therefore validate our implementation in the range of Reynlods numbers considered.



(a) Square Cylinder (b) Circular Cylinder

Figure 8.12: **Drag Coefficient of Square and Circular Cylinder**. Blue line represents results from literature, red symbols deal with numerical results of the present simulation. Red cross describe drag coefficients obtained using the correct rounded-shaped reference area, whereas red dots deal with the approximation of the shape in the CFD grid.

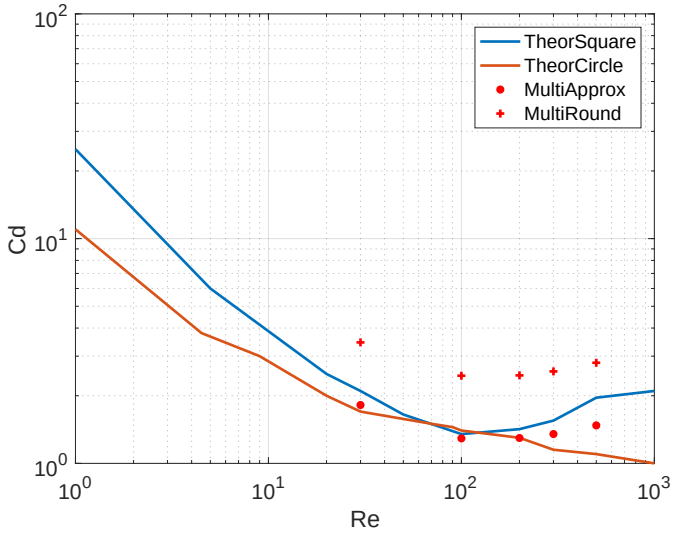


Figure 8.13: **Drag Coefficient of Cluster of Cylinders compared to literature data**. Blue and orange line represents results from literature (square and circle, respectively), red symbols deal with numerical results of the present simulation. Red cross describe drag coefficients obtained using the correct rounded-shaped reference area, whereas red dots deal with the approximation of the shape in the CFD grid.

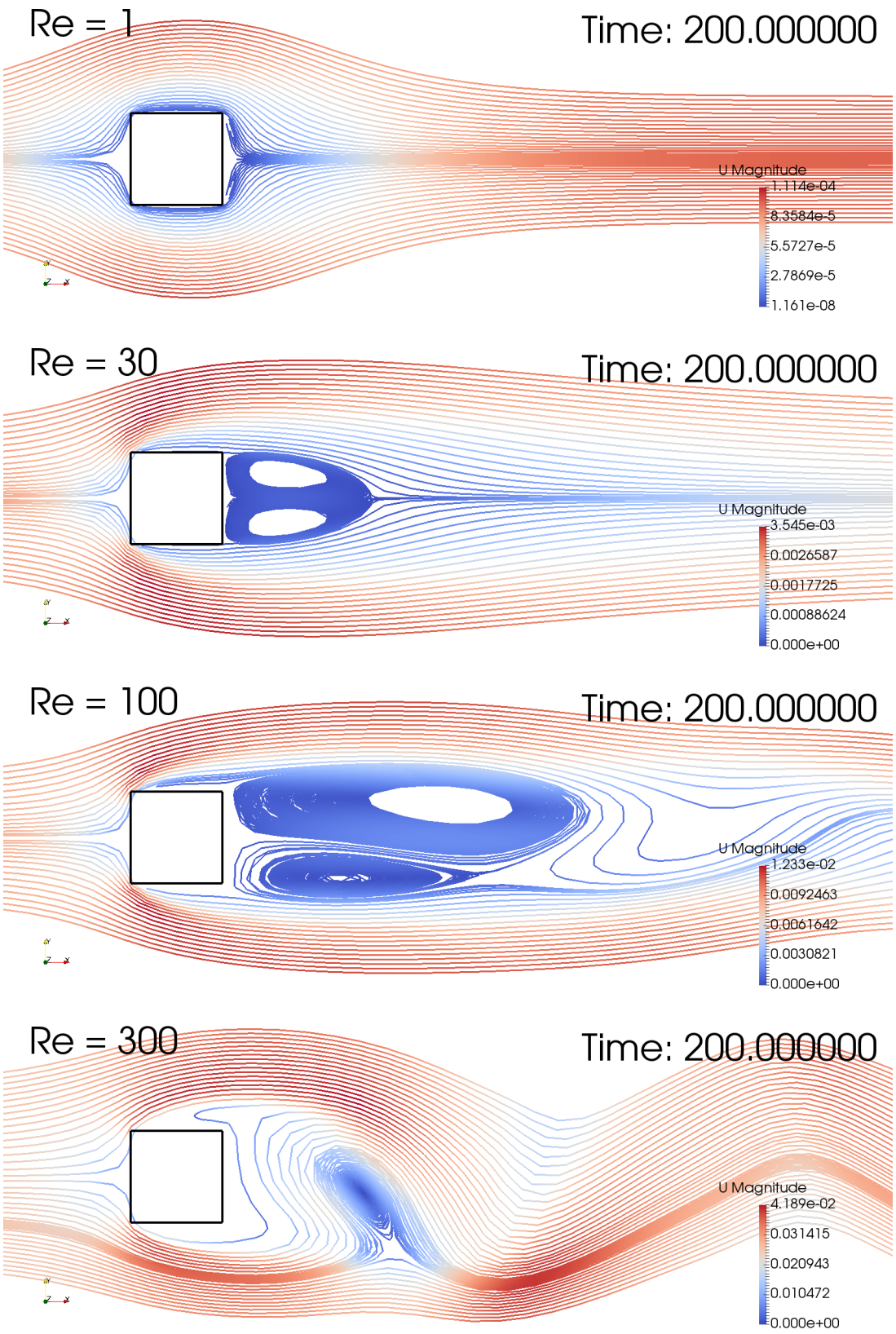


Figure 8.14: **Streamlines of flow around a square cylinder at different** *Re*.

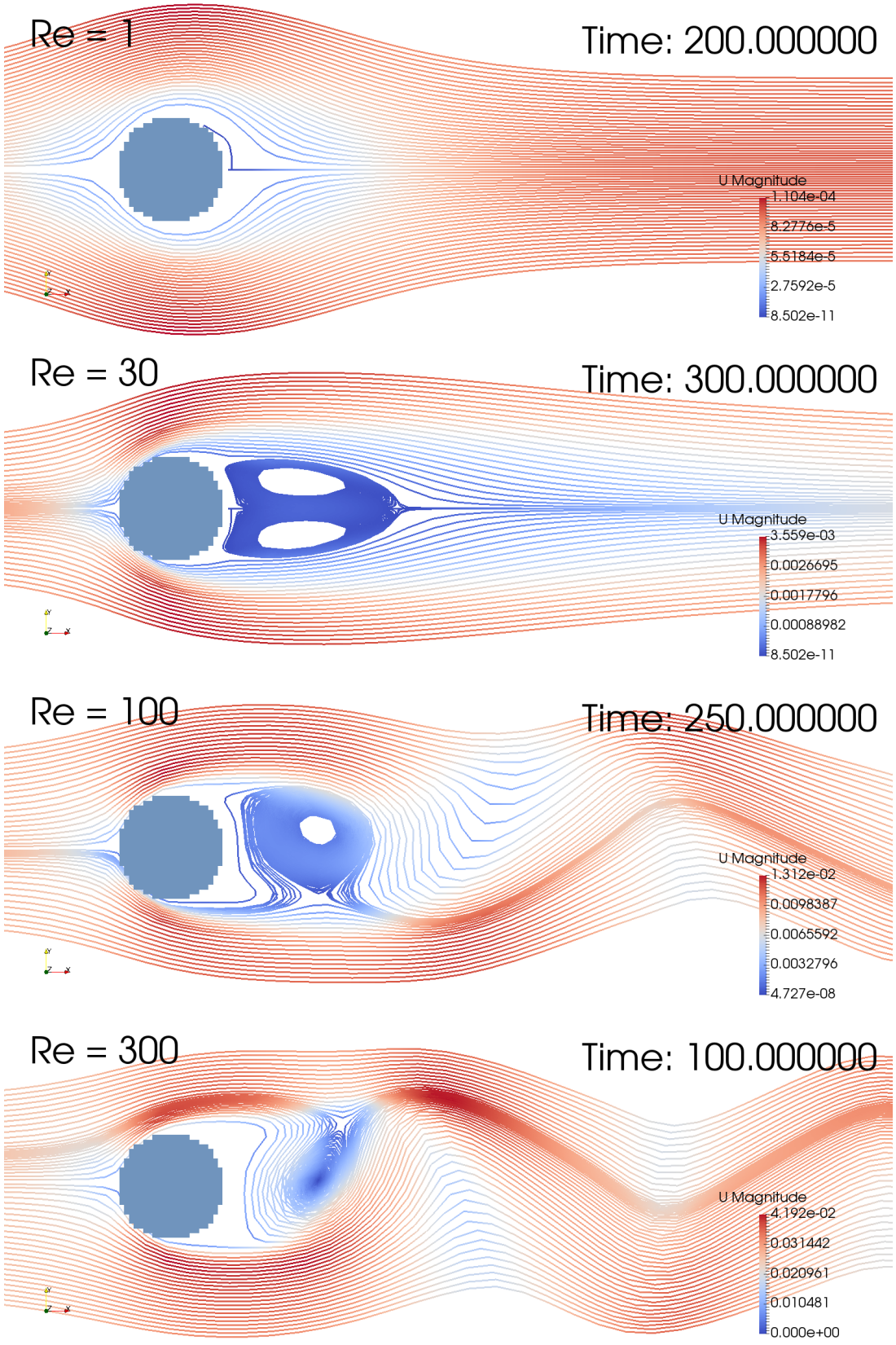


Figure 8.15: **Streamlines of flow around a circular cylinder at different** Re.

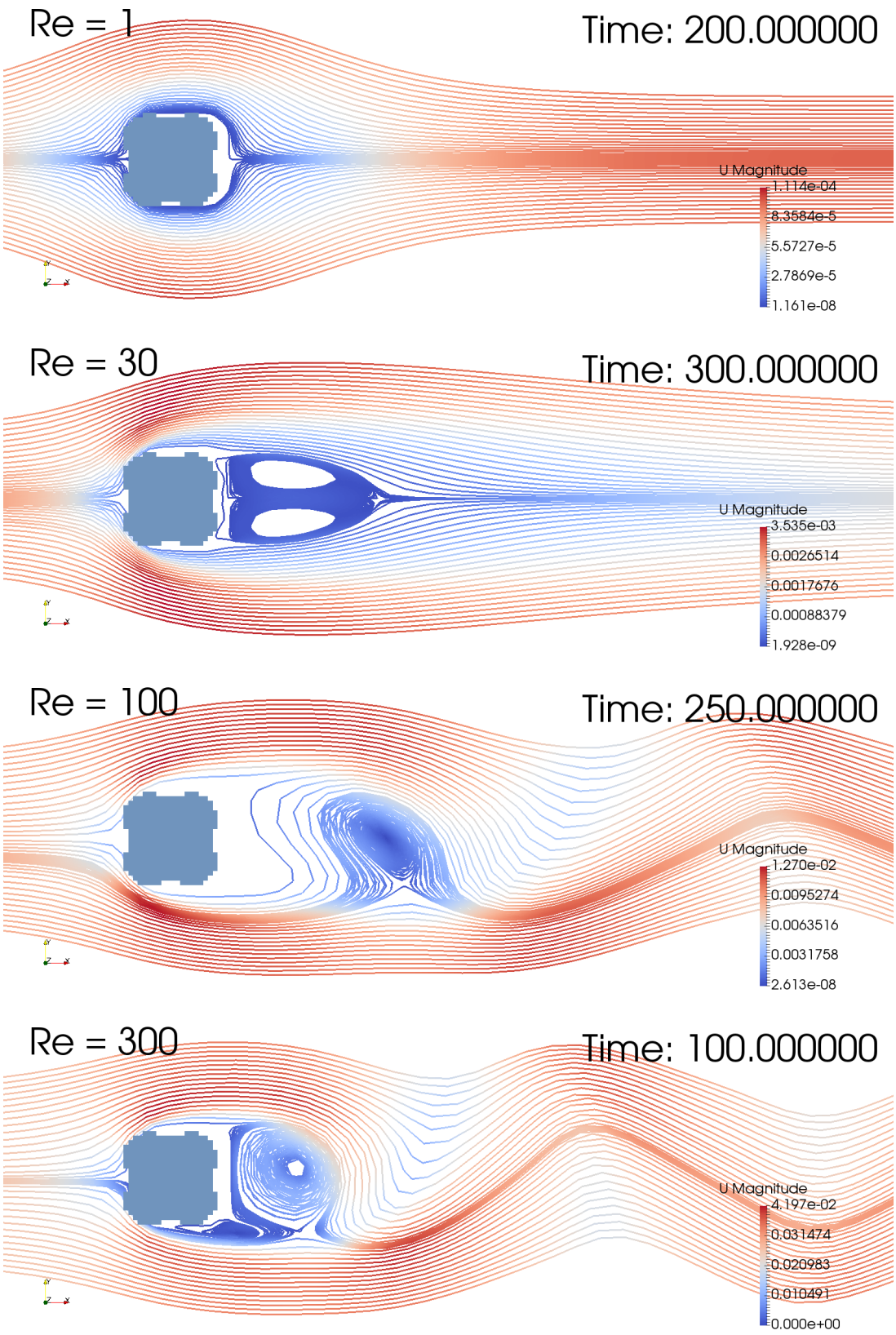


Figure 8.16: **Streamlines of flow around a cluster of circular cylinders at different** *Re*.

9 Applications

In this Chapter we propose two applications of the features developed and implemented during this project. In particular, we set up two cases that can be relevant to academic and industrial studies for validations and further studies and developments. We focus on the resolved coupling between non-spherical particles and a fluid.

The first case proposed is the *domino* fall of rectangular cylinders in water. the physical properties of the materials are arbitrarily chosen, but the relevance of this experiment is the possibility to measure easily the accuracy of the simulation by measuring the time necessary to the last item to fall. Finally, we propose a case to simulate the falling of rocks in a pipe to the sea bed. This kind of experiment is often used in marine and petroleum engineering to study the packing of rocks under the influence of the ocean current and the effects on pipe systems.

These are just two examples in which fluid-particles interactions play a fundamental role in the dynamics of a physical system. The variety of scenarios is extremely wide and this makes the topic of interaction between fluids and solids thrilling and thriving.

9.1. Domino in Water

An interesting scenario in which the interaction between fluid and particle becomes relevant is given by the simulation of a domino fall of objects immersed in water. Performing the simulation as pure particle-particle interaction we have results of falling particles in vacuum, whereas activating the fluid-particle solver we expect a slow down of the falling due to the influence of the fluid on the particle fall. An effective way to compare the numerical results with experiments is to measure the time in which the final object falls due to the *domino effect*.

The workflow followed in the setting up of the numerical experiment has been the following:

1. Fix material properties of the object and the box
2. Approximate the object with a cluster of spheres
3. Tune the damping coefficients to avoid instabilities and unphysical bouncing effects between the box and the objects
4. Perform pure particle simulation, taking under consideration the interactions between the objects with themselves and the walls
5. Trigger the influence of the fluid in the particles motion (and viceversa) with resolved coupling method
6. Perform different simulations varying parameters (CFD timestep, coupling time, mesh refinements, number of spheres per objects)

As a first step, we need to fix material properties for the particles and the fluid. As far as the fluid is concerned, we consider water, therefore the physical properties are well known and they are the same as the ones used in Chapter 8: ($\rho = 1000\text{kg}\,\text{m}^{-3}$, $\nu = 10^{-6}\text{m}^3\,\text{s}^{-1}$). As solid, we consider two different materials for the objects and the walls, in particular we choose rectangular cylinders of steel and a box of plexiglas. The physical properties are summed up in Table 9.1.

Property	**Value**
Particle density	$\rho = 8000\,\text{kg}\,\text{m}^{-3}$
Wall density	$\rho = 1180\,\text{kg}\,\text{m}^{-3}$
Young Modulus	$E = 3 \cdot 10^9\,\text{kg}\,\text{m}^{-1}\,\text{s}^{-2}$
Friction Coeff	$\mu = 0.4$
Normal damping Coeff	$\eta_n = 2 \cdot 10^5\,\text{m}^{-1}\,\text{s}^{-1}$
Tangential damping Coeff	$\eta_t = 1 \cdot 10^4\,\text{m}^{-1}\,\text{s}^{-1}$

Table 9.1: **Solid Properties**.

We decide to approximate the rectangular cylinder with 2 distinct clusters of spheres, with 20 and 40 spheres per cluster. In the case with 20 spheres, we use a layer of 4 spheres as section and we use 5 layers to build the box. In the case of 40 spheres, instead, we use 9 layers with the same distribution and we consider a skeleton of 4 spheres in the vertical direction to approximate better the shape of the horizontal edge. We show the clusters used in these cases in Figure 9.1.

(a) Coarse (20) (b) Fine (40)

Figure 9.1: **Clusters of spheres to approximate a Rectangular Object**.

In order to provoke the fall of the boxes, we give to the first box of the row an inclination of 12° with respect to the y axes. We show the initial configuration of the system in Figure 9.2.

Then, we choose the damping coefficients performing the numerical experiment more times and taking the threshold value that allows to detect collisions avoiding oscillations which do no happen in the real-life scenario with the same physical parameters. The damping coefficients (tuned for the case of 20 spheres in the cluster) are given in Table 9.1.

Afterwards, we consider the influence of the fluid in the system and we perform simulations of different scenarios. In particular, we consider the coarse and fine approximation of the rectangular objects and we analyze two different situations: complete influence between fluid and solid phase, named *TwoWay* and influence of the particle on the fluid dynamics and not vice-versa, named *OneWay*. For every case, we consider two different time steps for the CFD part (and therefore for the coupling time, since we exchange information every CFD step): we use $\Delta T_{CFD} = 0.001$s and $\Delta T_{CFD} = 0.002$s. For each of these cases, we study the time necessary for the last object of the row to fall. In Table 9.2 we report the falling times of the domino configuration in the different scenarios. We also add the execution time (in seconds s) of the simulations.

In order to achieve accurate results in a reasonable computation time, we activate the dynamic mesh refinement feature in OpenFOAM: this will cause a dynamic local mesh refinement and un-refinement influenced by the scalar field *interface*, which is evaluated from the *void fraction* scalar field applying some thresholds. To have comparable results, the update of the mesh is constant in every case and performed every 0.01 seconds.

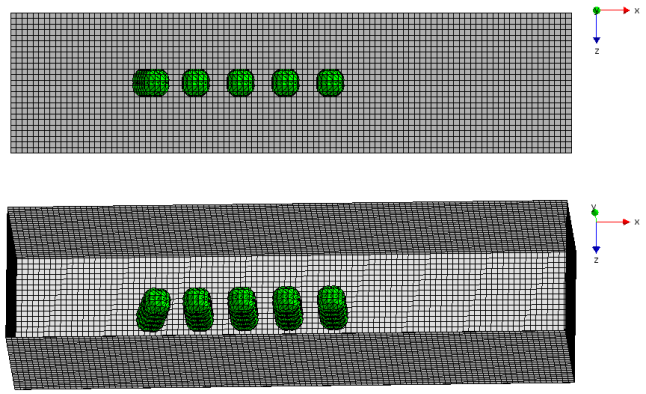


Figure 9.2: **Initial Configuration**. Top view and perspective view. The left object is given an initial angle of 12° w.r.t y axis to provoke the domino falling. Here, the coarse cluster is used.

A priori Considerations

From the setting of the simulations, we expect that results obtained varying only the CFD time step are characterized by very similar falling times. Due to numerical errors that may occur in the integrations, in the saving times of the files and in the precision of writing the numerical values, we tolerate a small discrepancies between the results.

Considering the different configurations in the coupling, we expect the falling times of the *OneWay* coupling to be smaller than the falling times of the cases where we consider the influence of the fluid in the particle motion. Since solid particles move in water, we expect the arise of drag and lift forces and we also consider the presence of a buoyancy force. In particular, the forces that the fluid exerts on the particles are evaluated through `ShirgaonkarIB` model in CFDEM and a buoyancy model in HADES, *ad hoc* developed for fully immersed objects. These forces should slow the particle motion in the *TwoWay* cases.

Finally, the differences between the two configurations of the approximation should not play a fundamental role, but we highlight that since the slice of the base is made of 4 spheres in the Coarse case and 5 spheres in the Fine case, differences may arise due to the presence of a different number of contacts.

Cluster	**Coupling**	ΔT_{CFD} [s]	**Falling Time** [s]	**Exec Time** [s]
Coarse	OneWay	0.001	1.68	4261.27
		0.002	1.60	2310.76
	TwoWay	0.001	1.72	4257.17
		0.002	1.68	2376.37
Fine	OneWay	0.001	1.58	5473.66
		0.002	1.54	2938.56
	TwoWay	0.001	0.92	6096.13
		0.002	1.30	3042.93

Table 9.2: **Falling Times**.

Comments

Numerical results have been obtained for the 8 cases described in the previous paragraph and in Figure 9.4 we give a snapshot of the simulation in a slice in the y direction. This helps to give an idea of the falling objects and to highlight the influence of the particles on the fluid in all cases and the influence of the fluid on the particle in the cases characterized by the *TwoWay* coupling. The falling times are reported in 9.2 and a visualization of the data is given in 9.3. It is straightforward that the results present some discrepancies with respect to the expected values, therefore a careful investigation needs to be carried out.

Firstly, we observe that for the Coarse case, results are not too far away from each other. As expected, the objects fall slower in the full coupling with respect to the case in which we neglect the influence of the fluid in the particle motion. This is reasonable due to the arise of the drag force that slow the fall of the objects. We also observe that using a smaller CFD time step slightly anticipates the fall. This is possible due to several and successive rounding errors in the communications between OpenFOAM and HADES: number of digits of precision, constraints on saving more files and more frequently are reasonable causes of this anticipation of the fall. Remind also that the drag forces are, of course, time dependent and when we are using a larger CFD time step we are also coupling the phases less frequently, causing larger error in the simulation. Finally, we emphasize that we are analyzing results with Paraview and we visualize data with an interval of 0.2 s, so this is the precision of our measurements. Nevertheless, taking all these facts into account, the falling times are not so far from each other and with the level of accuracy adopted in this simulation, we can be reasonably satisfied.

Instead, severe discrepancies appear in the Fine case. Let us firstly consider the *OneWay* configuration, where results are still in accordance with the ones of the Coarse case: the bigger the CFD time step, the faster the fall. We notice that the fall is slightly faster than the Coarse case. A possible explanation of this lies in the different geometrical configuration of the clusters in the two cases. In fact, in the Coarse case the base of the cluster is made of 4 spheres and this leads to 4 contact points with the wall, whereas in the Fine case 5 spheres compose the base of the objects. This leads to more contact points and probably the damping coefficient should be tuned again to obtain reliable results. In fact, the objects oscillate more and this lead to a less stable configuration, where the fall of the object is anticipated. This can play a role also in the arise of significant discrepancies of the Fine case with a full coupling. Contrary to the expectations, we observe that activating a full coupling to the system we cause an anticipation of the fall. This is counter-intuitive and we can not validate our case for this configuration. An explanation for this behavior can probably be again a damping coefficient not tuned, in particular lower than the necessary one. Adding a sphere in the center of the base, again, can lead to stronger oscillations which can be emphasized once we activate the full coupling of the system: the oscillation in velocity are transmitted to the fluid, which influence at the next CFD time step (= coupling time) the particle motion itself, with the arise of drag force, but more importantly, of lift force that will cause even more instabilities. Moreover, due to the buoyancy force, the solid will feel less gravity than before and this, again, can cause the need of more damping. Hence, an accurate tuning of the damping parameter is necessary to perform accurate simulations and its evaluation is one of the critical parts of a DEM (and CFD-DEM) model.

Even if in this case the damping coefficient seems to be the most relevant constraint to the accuracy of the simulation, we also remind that the coupling time is a critical parameter for the CFD-DEM coupling. With a too large coupling time, the particle motion can be unable to react to the fluid influence and simultaneously, the fluid can be unable to react to the presence of the particles in the correct positions and with the correct velocities. A careful study on the bounds for the coupling time will be therefore necessary in future and we will include this in the list of future research direction in the final Chapter of this Thesis.

In conclusion, results are close to the expected ones for the Coarse Case. In the Fine case, the *OneWay* coupling presents reasonable results, and the distance between this configuration and the correspondent one in the Coarse case gives the intuition that the reason for the discrepancies may lie in the damping coefficient tuning. Instead, the *TwoWay* configuration for the Fine case presents anomalies and we can not consider the results for this particular situation as reliable.

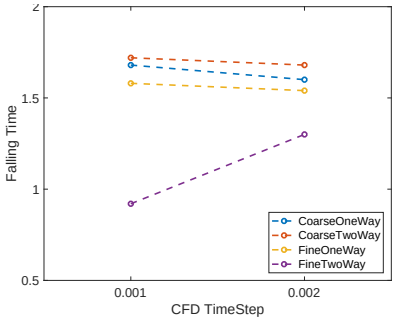
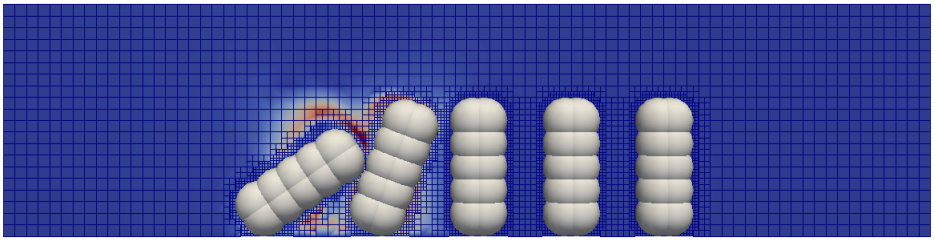
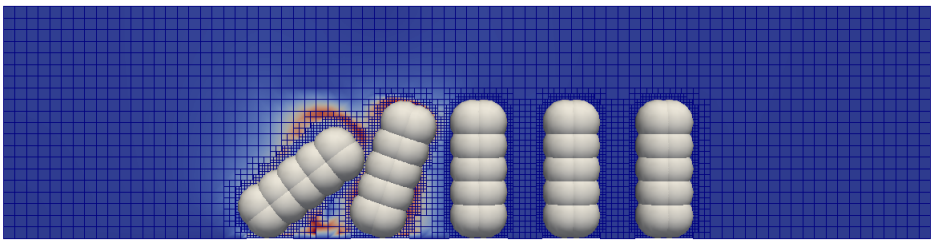


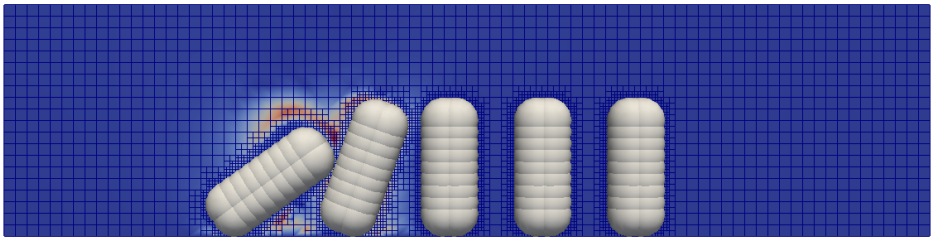
Figure 9.3: **Falling Times in different coupling configurations**.



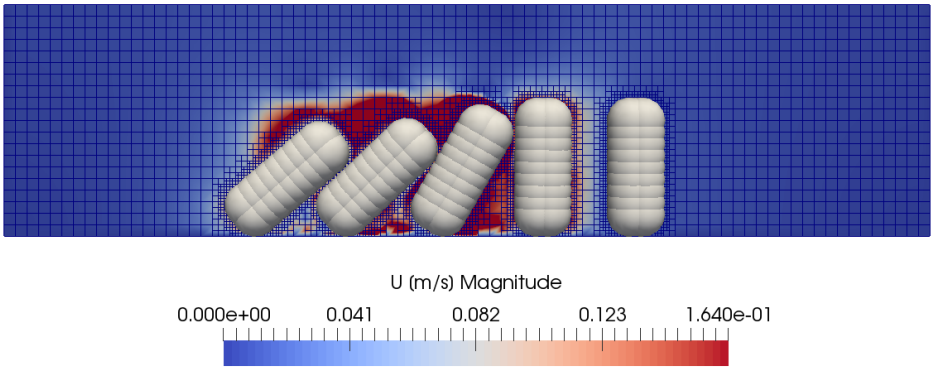
(a) Coarse. One Way



(b) Coarse. Two Way



(c) Fine. One Way



(d) Fine. Two Way

Figure 9.4: **Numerical Results at time** $t = 0.5$ **with** $\Delta T_{CFD} = 0.001$.

9.2. Falling Rocks on the Sea Bed

A typical scenario of geotechnical as well as oil and gas engineering is given by the fall of rocks, in a pipe, to the sea bed. In this Section we present some preliminary results of this case. A complete and deep study of the configuration was beyond the scope of the MSc project, but we chose this scenario to show the new capabilities of the solver and to give a taste of the possible applications of the research carried out during this project.

In order to achieve the simulations of falling rocks in water, we firstly need to set up the domain of computations both for the fluid and the particles. In particular, we consider a box of 4 m length (x direction), 3 m width (y direction) and 4 m height (z direction). In the center of the section, from $z = 2$m to $z = 4$m, we collocate a pipe with rectangular section of 1m $\times$ 0.5m. In Figure 9.5 we show the domain of computations as far as the fluid is concerned. As stated in the previous Chapters, the particle solver (HADES) has the limitation that only Hertzian contacts between particles and infinite planes can be considered. Therefore, due to the fact that the pipe ends in the middle of the domain, we could not build the pipe with planes. To overcome this limitation, we built the pipe with beds of spheres, with the appropriate size to maintain the width of the pipe. The domain for the particle evaluations is shown in 9.6.

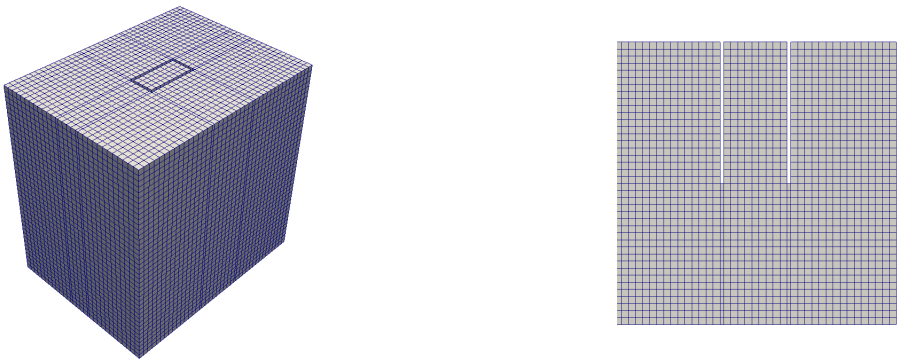


Figure 9.5: **CFD Domain**

Figure 9.6: **DEM Domain**

After the set up of the domain, we need an approximation of the rock with a cluster of particles. We decide to use the cluster shown in 9.7 and given by the following parameters:

```
centerX = [ -0.03,  0.00,  0.03,  0.03 ];
centerY = [ -0.01,  0.00,  0.03,  0.03 ];
centerZ = [  0.00,  0.00,  0.01, -0.01 ];
radii   = [  0.04,  0.04,  0.03,  0.025];
```

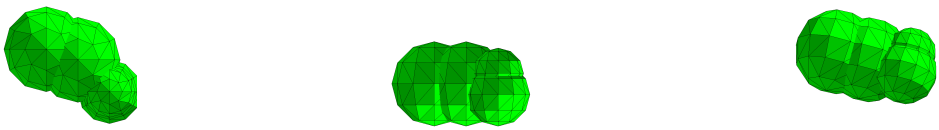


Figure 9.7: **Rock Cluster**

The rocks fall in water, so the fluid has the standar properties for water ($\rho = 1000\text{kg}\,\text{m}^{-3}$, $\nu = 10^{-6}\text{m}^3\,\text{s}^{-1}$). The parameters selected for the rocks are shown in Table 9.3. We remind that the damping coefficients are critical parameters which would require further and research to validate the solver. In this case, we estimate their value avoiding nonphysical oscillations in the results, but the accuracy of the simulation would require a better tuning.

Property	**Value**
Particle density	$\rho = 1180\,\text{kg}\,\text{m}^{-3}$
Wall density (water)	$\rho = 1000\,\text{kg}\,\text{m}^{-3}$
Young Modulus	$E = 4.5 \cdot 10^9\,\text{kg}\,\text{m}^{-1}\,\text{s}^{-2}$
Poisson Ratio	$\lambda = 0.35$
Friction Coeff	$\mu_{fr} = 0.6$
Rolling Friction Coeff	$\mu_{roll} = 0.001$
Normal damping Coeff	$\eta_n = 9 \cdot 10^5\,\text{m}^{-1}\,\text{s}^{-1}$
Tangential damping Coeff	$\eta_t = 3 \cdot 10^5\,\text{m}^{-1}\,\text{s}^{-1}$

Table 9.3: **Solid Properties**.

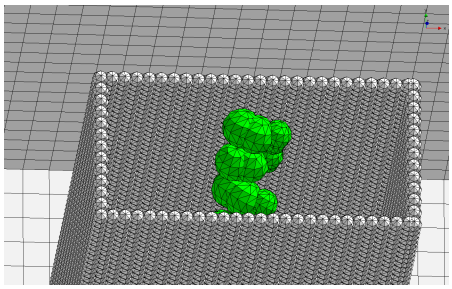


Figure 9.8: **Rock injection** One rock is inserted every 0.1s.

Now we decide how to insert the rocks in the simulation. In particular, we insert one rock every 0.1s in the top of the domain and we assign to each rock a velocity of $2\text{m}.s^{-1}$ in the negative z direction. To avoid unrealistic results, each rock is given a random initial orientation. In Figure 9.8 we give a snapshot of the injection.

In the system we consider the following forces:

- Gravity
- Buoyancy
  Since the rocks are fully immersed and the fluid is considered incompressible, we evaluate the buoyancy effect directly in the particle solver, using as input the (constant) density of the fluid.

We firstly run the simulation as pure DEM system until $t = 60\text{s}$. In Figure 9.9 we show the final configuration reached at $t = 60\text{s}$. The computation time is approximately 3 hours. For visualization purposes, the geometry of the domain is omitted.

We now consider the presence of the fluid in the system. Due to time constraints in the project, we consider a One Way configuration: we analyze the influence of the particles in the fluid, but not vice-versa. In particular, we consider the buoyancy contribution, directly on the particle solver, but not the drag component of the force. Hence, no data arising from the fluid is read by HADES in this situation, since the density is constant and fixed *a priori* in the Buoyancy model in HADES. The fluid-particle configuration is more demanding from a computational point of view, hence we run the simulation up to 15s. Using 6 processors, the computational time was approximately 5 days. This shows that the major contribution to the computational resources was given by the CFD solver and the communication between the CFD and the DEM solver: the particle simulation by itself was relatively negligible in this configuration. Hence, the most expensive phases from the computational point of view are: the location of the particles, the evaluation of the scalar fields that deal with

particles (voidfraction, interface), the dynamic mesh refinement performed every CFD step, the CFD solution and the file communication.

In Figure 9.10 we show a slice of the CFD domain and the packing of rocks reached at $t = 10$s. The influence of the presence of the rocks in water leads to a situation similar to a jet flow. In Figure 9.11 we show the velocity field of water and the packed structure of rocks in different times, whereas Figure 9.12 deals with the approximation of rocks inside the CFD mesh: the scalar field of void-fraction is given.

We remind that this final case had the purpose to give the reader a taste of the implementation described in this Thesis. Hence, the accuracy of the simulation was not tested, but we showed one very interesting situation in which fluid-particle simulation can be used to optimize work performances, to design industrial products and to innovate current technologies in several branches of engineering.

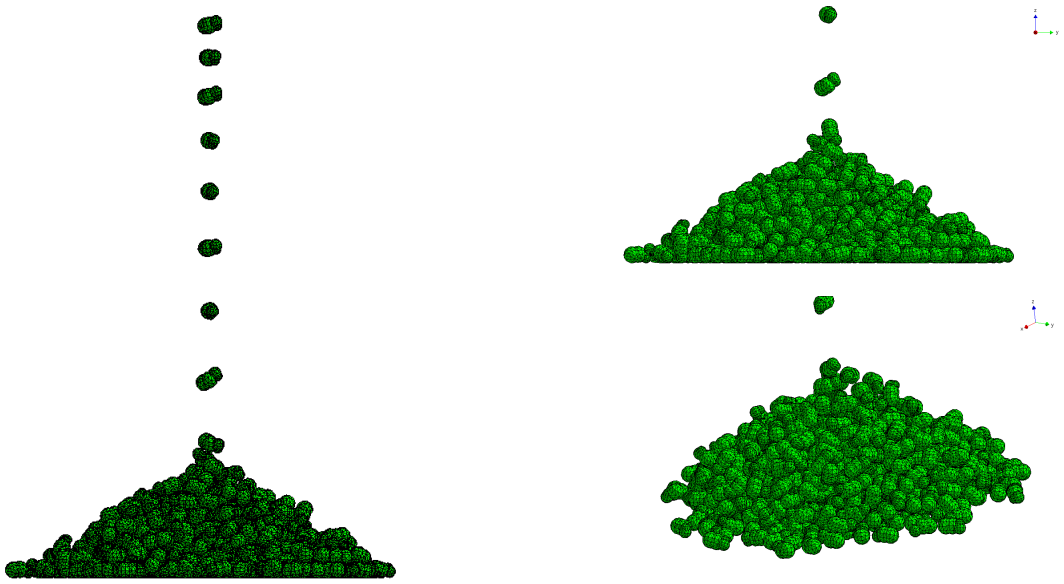


Figure 9.9: **Pure DEM Packing at** $t = 60$s.

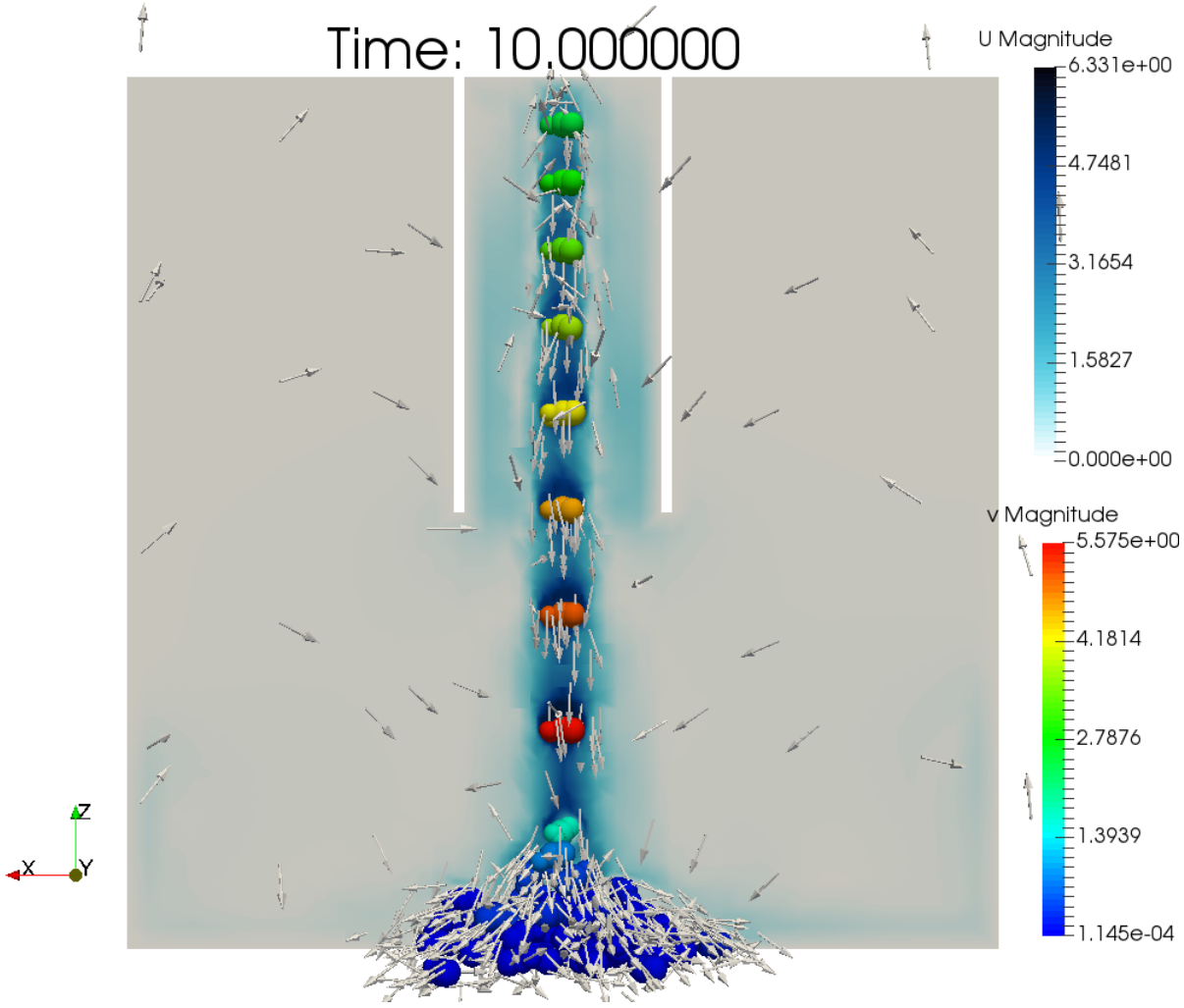


Figure 9.10: **Velocity of the fluid and of the rocks at time** $t = 10$s**.** In the Blue Scale, velocity of the water is given in a central slice of the domain (plane x, z). In the rainbow scale, the velocity of the rocks is plotted. Particles are subjected to gravity and buoyancy force. The arrows represent the local velocity of the fluid.

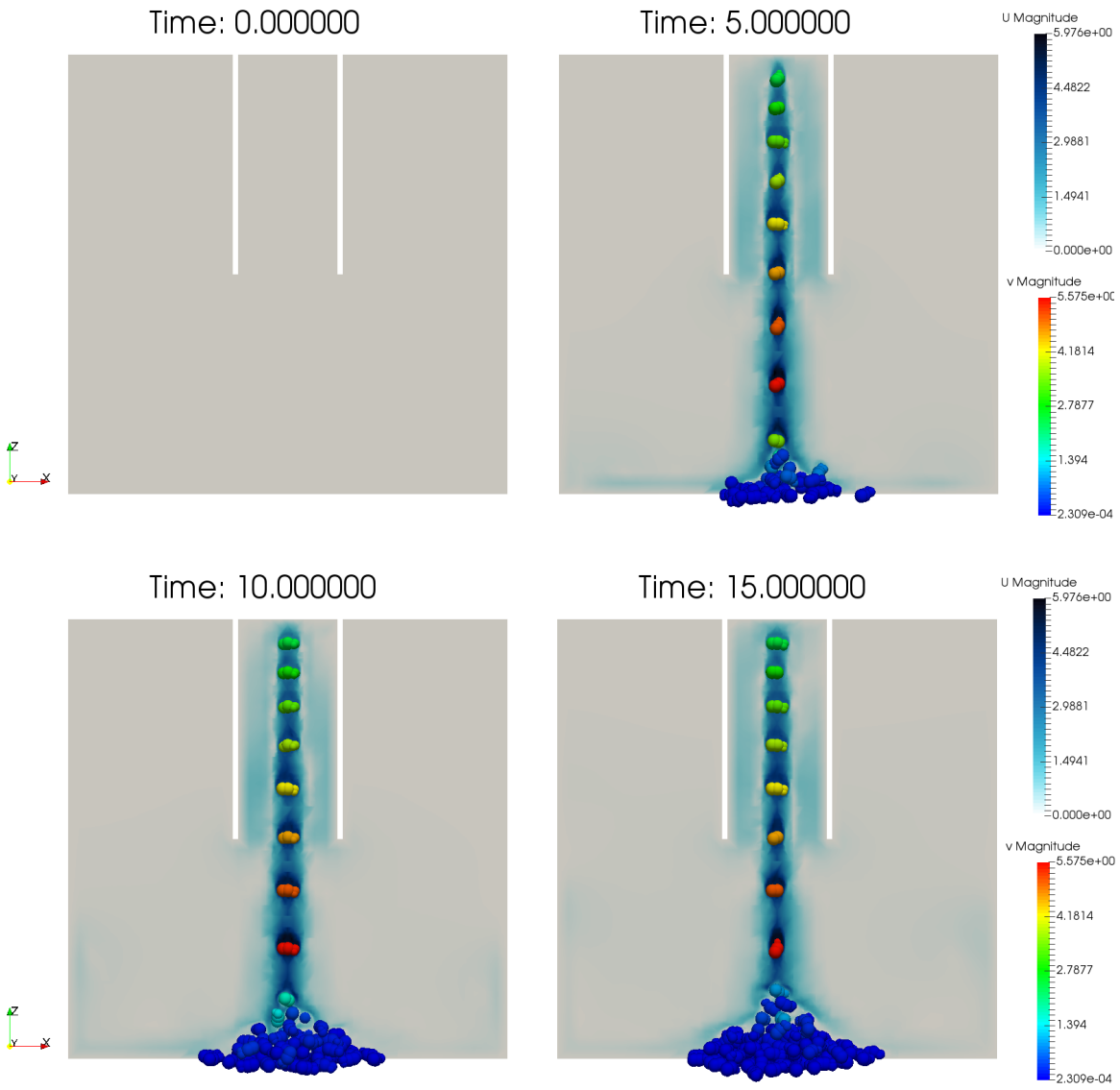


Figure 9.11: **Velocity Field**. In the Blue Scale, velocity of the water is given in a central slice of the domain (plane x, z). In the rainbow scale, the velocity of the rocks is plotted. Particles are subjected to gravity and buoyancy force.

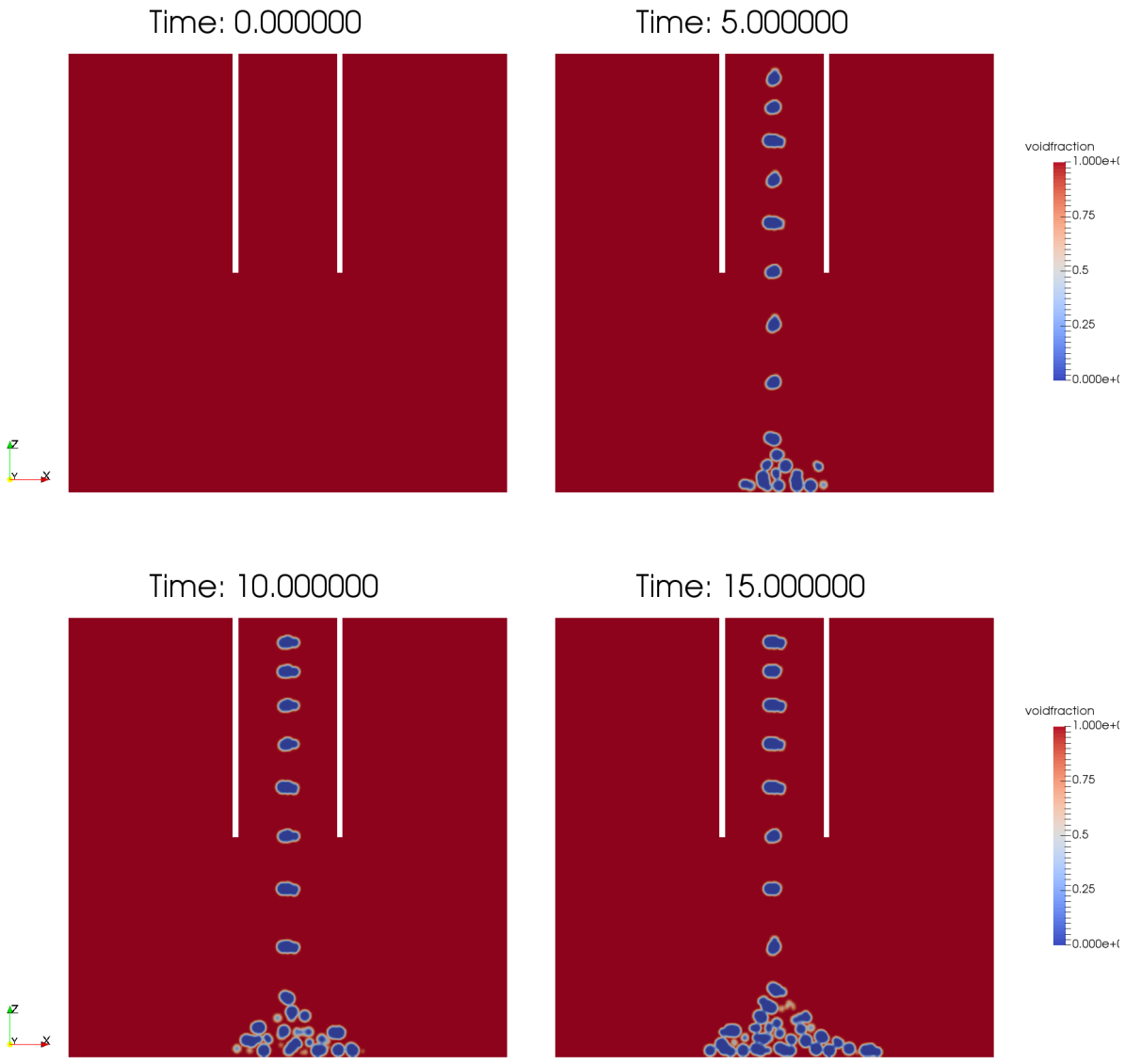


Figure 9.12: **Void-fraction Field**. The scalar void-fraction field used to locate the particles in the CFD mesh is given for different times.

Conclusions and Future Directions

The aim of this project was to study the state of the art of the methods developed for fluid-particles simulations and possibly implement new features in open-source software available to industries. To this purpose, we dedicated the first part of this Thesis report to a review of the techniques available in literature. This theoretical part contains the basics of the Discrete Element Method and the different approaches that can be used for the coupling. In particular, we distinguished between resolved and unresolved coupling, depending on the relative size of the particles involved and the CFD cells. We discussed these two possible configurations and we gave some details on the possible approaches to adopt.

Afterwards, we decided to proceed in the project to answer to the research questions described in the final Chapter on the Theoretical Background part. In particular, we analyzed the Verlet algorithm that is the common integrator implemented in particle solvers and we found out that it is an example of a second-order symplectic partitioned Runge Kutta methods. Therefore, since it is in the intersection of the theory of symplecticity and partitioned Runge Kutta methods, we state that further research for an higher-order integrator within these classes may improve convergence behaviour of the numerical simulations. Then, in order to have the possibility to use the innovative integration implementation of the Verlet algorithm with a non-fixed time step in HADES, we focused on the coupling of OpenFOAM with HADES, exploiting the CFDEM framework and substituting LIGGGHTS with this particle solver. Details of the code development are given in the proper Section. A further research question dealt with the limitation of applicability of DEM to spherical particles and this constraint is particularly severe in resolved cases. Therefore, we analyzed the possibilities in literature to overcome to this drawback. Two alternatives were presented and discussed: superquadric approximations and the multisphere approach. After a careful discussion about pros and cons of this techniques, we chose to implement in HADES the multisphere method and we described the work-flow adopted in detail.

In the final part of the thesis we tested the new-implemented features in benchmark problems and we discussed their accuracy and their performances. In particular, the test of filling and discharging of cubes approximated with clusters of spheres showed good initial behaviour, but discrepancies in the evolution of the simulation in time. This can be explained with the fact that the case studied belonged to the *worst case scenarios* for the multisphere approach, since the approximation of extremely stable and regular geometrical shapes is for sure not optimal for the validation of a multisphere solver. Tests with more rounded geometrical shape will be necessary to validate the feature implemented in cases where the multisphere approach is a powerful resource. Then, we tested the CFD-DEM framework studying the flow around a square cylinder and results were satisfying with respect to the literature data available. Two applications were described in the final Chapter: the fall of rectangular objects in water and the fall of rocks in the sea bed. A small study on the influence of the CFD time step (and consequently of the coupling time) was carried out in the first case, whereas due to time constraints on the project only preliminary results could be obtained for the fall of rocks.

During the development of the project and in the test phase, some further research questions arose and some ideas on necessary improvements in the current implementation emerged but could not be carried out due to the limited time available in the framework of an MSc Thesis project.

Theoretical and Experimental research

In particular, a lack of literature data and theoretical research was found out for these two topics:

- **Tuning of damping coefficients in DEM**
  As stated in Chapter 2 and 9, models for the damping coefficient are quite primitive in literature and usually the damping coefficient is tuned by the user to achieve results in accordance with physical models. Wrong damping coefficients can lead to unstable DEM simulations and the influence of the fluid in a CFD-DEM system can emphasize these instabilities. Hence, models for the damping should be developed both experimentally and numerically to improve stability and accuracy of the simulations.

- **Coupling time for CFD-DEM**
  Due to the fact that, until some years ago, the computational resources were simply not enough to perform relevant simulations, not enough models are available to set upper or lower bounds on the coupling times between the CFD and DEM evaluations. In the current implementation, the coupling time needs to be a multiple of the CFD time step. This leads to the following two extremes: if the coupling time is too small, excessive computational time is required, whereas if the coupling time is too large, the two phases do not manage to react to each other and the simulation results are not reliable anymore. Models that link physical behaviours to numerical quantities (CFD time steps, coupling times, ...) should be developed and researched, to avoid problem-specific approaches.

Test of the current implementation

As stated above, the multisphere approach has been implemented and tested in falling and discharge of cubes. Unfortunately, the cube is an extremely regular and stable geometrical object and its approximation with a cluster of spheres leads to intrinsic instabilities due to the roundness of the faces. Therefore, a further test would be necessary:

- **DEM Multisphere for rounded shapes**
  Data are available in literature for the packing of ellipsoids, corn-shaped particles or pharmaceutical pills. An approximation with clusters of spheres of these shapes could lead to relevant and interesting results, since the objects present rounded-shapes and the multisphere approach would not introduce artificial instabilities in the shapes.

Further implementations

During the project, some ideas on possible improvements were developed, but due to limited time they could not be implemented. We now give a list of possible improvements of the current implementation.

- **pimple algorithm in CFDEM**
  The current PISO routine does not allow an adjustable CFD time step: it has to be fixed *a priori*. This is of course not optimal since there is the possibility that a bigger ΔT_{CFD} could be used for relevant time intervals. The PIMPLE routine is an efficient alternative and improvement of PISO and it allows ΔT_{CFD} to be varied in time fulfilling some constraints. Since the pure PIMPLE algorithm is given in the source code of OpenFOAM, it would be worthy to adapt it to the CFDEM framework. We have to remind that bounds on ΔT_{CFD} will have to be imposed to achieve meaningful results between the coupling of CFD and DEM. The constraint will be important since the coupling time is equal to ΔT_{CFD}. Hence, a too large ΔT_{CFD} would cause a bad coupling between the two phases.

- **Possibility of simultaneous resolved-unresolved coupling**
  The resolved and unresolved coupling approaches are performed through the assignation of particles to the correct cloud (`cfdemCloud` for unresolved and`cfdemCloudIB` for resolved). A simultaneous handling of the two clouds is possible through a combination of the solvers available. The critical phase will be to assign the particle to the correct cloud (small to unresolved and big to resolved). It should not cost too much effort to handle this assignation *a priori* through some conditions on the name of the output files saved by HADES. The small particles should impose point forces on the equations and then the immersed boundary routine should be performed, as proposed in [24].

- **Possibility to delete clusters in HADES**
  In the current implementation, due to the presence of a mapping between the ID of the particles in the clusters and the ghost particles which are used in the integration, particles injected cannot leave the domain and therefore they cannot be deleted during a simulation. This constraint should be solved with a more robust mapping of particle IDs.

- **Possibility to insert particles of different clusters in fluid-particle simulations**
  Now, only one kind of cluster of particles can be handled by HADES. Different clusters can be generated, the motion of different clusters can be performed as rigid bodies, but the contact model is said to evaluate contacts in an efficient way (neglecting contacts between spheres of the same clusters) only with one kind of clusters. A generalization of the procedure should be not difficult.

- **Generalization of Hertzian Contact Models in HADES**
  Currently, Hertzian contact models are available only in two very simple configurations: sphere-sphere and sphere-infinite plane. A generalization of the contact models will be necessary to consider complex industrial scenarios, with geometries form STL files. Models for handling vertex, edge and face contacts should be studied and implemented.
- **Implementation of a new integration algorithm**
  Higher order algorithms may be implemented instead of the Verlet procedure. Symplectic partitioned Runge Kutta Methods provide an alternative framework for the development of higher order integrators.

In Conclusion, after a detailed literature study on the current models and methods for the coupling between CFD and DEM for fluid-particle simulations, in this project we propose a strategy for the development of more performing numerical integration, we allow to perform simulations with a coupling between OpenFOAM and HADES, exploiting the feature in HADES that allows to have a not-fixed time step and, more importantly, we allow the coupling between a fluid and non-spherical particles.

The field of numerical simulations is extremely dynamic and it is now living a golden era on innovative methods and strategies to perform efficient and reliable simulations of industrial and academic problems. In this framework, we consider this project as a feasibility study, adopting strategies to generalizing the current implementations adding new features, but we have to be aware that a lot is still to be done and deeper researches will open the gates to scenarios that, even now, may seem unrealistic.

Bibliography

[1] M. Afkhami, A. Hassanpour, M. Fairweather, and D.O. Njobuenwu. Fully coupled les-dem of particle interaction and agglomeration in a turbulent channel flow. *Computers & Chemical Engineering*, 78:24 – 38, 2015. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2015.04.003.

[2] Falah Alobaid and Bernd Epple. Improvement, validation and application of cfd/dem model to dense gas–solid flow in a fluidized bed. *Particuology*, 11(5):514 – 526, 2013. ISSN 1674-2001. doi: https://doi.org/10.1016/j.partic.2012.05.008. URL http://www.sciencedirect.com/science/article/pii/S167420011200171X.

[3] Stefan Amberger, Michael Friedl, Christoph Goniva, Stefan Pirker, and Christoph Kloss. Approximation of objects by spheres for multisphere simulations in dem. In *ECCOMAS 2012 - European Congress on Computational Methods in Applied Sciences and Engineering, e-Book Full Papers*, 09 2012.

[4] John D. Jr Anderson. *Fundamentals of Aerodynamics*. McGraw-Hill, Shoppenhangers Road, Maidenhead, Berkshire, SL6 2QL, UK, fifth edition edition, 2011. ISBN 978-007-128908-5.

[5] A. H. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981. ISSN 0272-1716. doi: 10.1109/MCG.1981.1673799.

[6] Bruno Blais, Manon Lassaigne, Christoph Goniva, Louis Fradette, and François Bertrand. A semi-implicit immersed boundary method and its application to viscous mixing. *Computers & Chemical Engineering*, 85:136 – 146, 2016. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2015.10.019.

[7] M. Breuer, J. Bernsdorf, T. Zeiser, and F. Durst. Accurate computations of the laminar flow past a square cylinder based on two different methods: lattice-boltzmann and finite-volume. *International Journal of Heat and Fluid Flow*, 21(2):186 – 196, 2000. ISSN 0142-727X. doi: https://doi.org/10.1016/S0142-727X(99)00081-8. URL http://www.sciencedirect.com/science/article/pii/S0142727X99000818.

[8] Alice Jordam Caserta, Hélio A. Navarro, and Luben Cabezas-Gómez. Damping coefficient and contact duration relations for continuous nonlinear spring-dashpot contact model in dem. *Powder Technology*, 302:462 – 479, 2016. ISSN 0032-5910. doi: https://doi.org/10.1016/j.powtec.2016.07.032. URL http://www.sciencedirect.com/science/article/pii/S0032591016304260.

[9] DCS Computing. Cfdem®coupling documentation. https://www.cfdem.com/media/CFDEM/docu/CFDEMcoupling_Manual.html, 2018. Accessed: 24.10.2018.

[10] DCS Computing. Liggghts(r)-public documentation, version 3.x. https://www.cfdem.com/media/DEM/docu/Manual.html, 2018. Accessed: 24.10.2018.

[11] C.T Crowe, J.D. Schwarzkopf, M. Sommerfeld, and Y. Tsuji. *Multiphase Flows with Droplets and Particles*. CRC Press, second edition edition, 2011.

[12] Kang Fang and Mengzhao Qin. *Symplectic Geometric Algorithms for Hamiltonian Systems*. Springer-Verlag, Berlin, Heidelberg, 2010. ISBN 978-3-642-01777-3.

[13] J. H. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics*. Springer, first edition edition, 2002. doi: 10.1007/978-3-642-56026-2.

[14] The OpenFOAM Foundation. Openfoam user guide, version 5.0. https://cfd.direct/openfoam/user-guide-v5/, 2017. Accessed: 24.10.2018.

[15] Feras Y. Fraige, Paul A. Langston, and George Z. Chen. Distinct element modelling of cubic particle packing and flow. *Powder Technology*, 186(3):224 – 240, 2008. ISSN 0032-5910. doi: https://doi.org/10.1016/j.powtec.2007.12.009. URL http://www.sciencedirect.com/science/article/pii/S0032591007006407.

[16] A. Goniva, C. Kloss, N.G. Deen, J.A.M. Kuipers, and S. Pirker. Influence of rolling friction on single spout fluidized bed simulation. *Particuology*, 10(5):582–591, 2012. ISSN 1674-2001. doi: 10.1016/j.partic.2012.05.002.

[17] C. Goniva, B. Blais, S. Radl, and C. Kloss. Open source cfd-dem modeling for particle-based processes. In *Proceedings of Eleventh International Conference on CFD in the Minerals and Process Industries*, 12.2015.

[18] A. Hager. *CFD-DEM on Multiple Scales - An Extensive Investigation of Particle-Fluid Interactions*. PhD thesis, Johannes Kepler Universität Linz, 2011.

[19] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 978-3-642-05220-0. doi: 10.1007/978-3-642-05221-7.

[20] Ernst Hairer, Nørsett Syvert P., and Gerhard Wanner. *Solving Ordinary Differential Equations I - Non-stiff Problems*. Springer-Verlag, Berlin, Heidelberg, 1993. ISBN 978-3-642-08158-3. doi: 10.1007/978-3-540-78862-1.

[21] Ernst Hairer, Christian Lubich, and Gerhard Wanner. Geometric numerical integration illustrated by the störmer–verlet method. *Acta Numerica*, 12:399–450, 2003. doi: 10.1017/S0962492902000144.

[22] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 978-3-540-30663-4. doi: 10.1007/3-540-30666-8.

[23] W. R. Hamilton. On quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science (3rd Series)*, xxv-xxxvi, 1844-1850.

[24] Yi. He, Andrew E. Bayly, and Ali Hassanpour. Coupling cfd-dem with dynamic meshing: A new approach for fluid-structure interaction in particle-fluid flows. *Powder Technology*, 325:620 – 631, 2018. ISSN 0032-5910. doi: https://doi.org/10.1016/j.powtec.2017.11.045.

[25] C. Kloss, C. Goniva, A. Hager, S. Amberger, and S. Pirker. Model, algorithm and validation for opensource dem and cfd-dem. *Progress in Computational Fluid Dynamics*, 12(2-3):140–152, 2012. doi: https://doi.org/10.1504/PCFD.2012.047457.

[26] H. Kruggel-Emden, E. Simsek, S. Rickelt, S. Wirtz, and V. Scherer. Review and extension of normal force models for the discrete element method. *Powder Technology*, 171(3):157 – 173, 2007. ISSN 0032-5910. doi: https://doi.org/10.1016/j.powtec.2006.10.004. URL http://www.sciencedirect.com/science/article/pii/S0032591006004360.

[27] H. Kruggel-Emden, S. Rickelt, S. Wirtz, and V. Scherer. A study on the validity of the multi-sphere discrete element method. *Powder Technology*, 188(2):153 – 165, 2008. ISSN 0032-5910. doi: https://doi.org/10.1016/j.powtec.2008.04.037. URL http://www.sciencedirect.com/science/article/pii/S0032591008002143.

[28] P. K. Kundu, I. M. Cohen, and D. R. Dowling. *Fluid Mechanics*. Academic Press, sixth edition edition, 2015.

[29] T.Y. Lam. Hamilton's quaternions. volume 3 of *Handbook of Algebra*, pages 429 – 454. North-Holland, 2003. doi: https://doi.org/10.1016/S1570-7954(03)80068-2. URL http://www.sciencedirect.com/science/article/pii/S1570795403800682.

[30] Guoping Li and Joseph A. C. Humphrey. Numerical modelling of confined flow past a cylinder of square cross-section at various orientations. *International Journal for Numerical Methods in Fluids*, 20(11):1215–1236, 1995. doi: 10.1002/fld.1650201103. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.1650201103.

[31] Wan-Qing Li, Tang Ying, Wan Jian, and Dong-Jin Yu. Comparison research on the neighbor list algorithms: Verlet table and linked-cell. *Computer Physics Communications*, 181(10):1682 – 1686, 2010. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2010.06.005.

[32] G. Lu, J.R. Third, and C.R. Müller. Discrete element models for non-spherical particle systems: From theoretical developments to applications. *Chemical Engineering Science*, 127:425 – 465, 2015. ISSN 0009-2509. doi: https://doi.org/10.1016/j.ces.2014.11.050. URL http://www.sciencedirect.com/science/article/pii/S0009250914007040.

[33] J. R. Miller. Analysis of quadric-surface-based solid models. *IEEE Computer Graphics and Applications*, 8(1):28–42, 1988. ISSN 0272-1716. doi: 10.1109/38.488.

[34] Zemin Ning and Mojtaba Ghadiri. Distinct element analysis of attrition of granular solids under shear deformation. *Chemical Engineering Science*, 61(18):5991 – 6001, 2006. ISSN 0009-2509. doi: https://doi.org/10.1016/j.ces.2006.03.056.

[35] Ronald L. Panton. *Incompressible Flow*. Wiley, forth edition edition, 2013. ISBN 978-1-118-41573-3.

[36] Alexander Podlozhnyuk, Stefan Pirker, and Christoph Kloss. Efficient implementation of superquadric particles in discrete element method within an open-source framework. *Computational Particle Mechanics*, 4(1):101–118, 2017. ISSN 2196-4386. doi: 10.1007/s40571-016-0131-6. URL https://doi.org/10.1007/s40571-016-0131-6.

[37] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992. ISBN 0-521-43108-5.

[38] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-34658-6. doi: 10.1007/b98885.

[39] F. Radjai and F. Dubois. *Discrete-element Modeling of Granular Materials*. Wiley, first edition edition, 2011.

[40] Soltanbeigi, Behzad, Podlozhnyuk, Alexander, Ooi, Jin Y., Kloss, Christoph, and Papanicolopulos, Stefanos-Aldo. Comparison of multi-sphere and superquadric particle representation for modelling shearing and flow characteristics of granular assemblies. *EPJ Web Conf.*, 140:06015, 2017. doi: 10.1051/epjconf/201714006015. URL https://doi.org/10.1051/epjconf/201714006015.

[41] Rui Sun and Heng Xiao. Diffusion-based coarse graining in hybrid continuum-discrete solvers: Applications in cfd-dem. *International Journal of Multiphase Flow*, 72:233 – 247, 2015. ISSN 0301-9322. doi: https://doi.org/10.1016/j.ijmultiphaseflow.2015.02.014.

[42] Rui Sun and Heng Xiao. Diffusion-based coarse graining in hybrid continuum-discrete solvers: Theoretical formulation and a priori tests. *International Journal of Multiphase Flow*, 77:142 – 157, 2015. ISSN 0301-9322. doi: https://doi.org/10.1016/j.ijmultiphaseflow.2015.08.014.

[43] Rui Sun and Heng Xiao. Sedifoam: A general-purpose, open-source cfd-dem solver for particle-laden flow with emphasis on sediment transport. *Computers & Geosciences*, 89:207 – 219, 2016. ISSN 0098-3004. doi: https://doi.org/10.1016/j.cageo.2016.01.011.

[44] Xiaomei Wang, Jianqun Yu, Fengyan Lv, Yang Wang, and Hong Fu. A multi-sphere based modelling method for maize grain assemblies. *Advanced Powder Technology*, 28(2):584 – 595, 2017. ISSN 0921-8831. doi: https://doi.org/10.1016/j.apt.2016.10.027. URL http://www.sciencedirect.com/science/article/pii/S092188311630348X.

[45] P. Wesseling. *Principles of Computational Fluid Dynamics*. Springer, first edition edition, 2001. doi: 10.1007/978-3-642-05146-3.

[46] H.N. Yow, M.J. Pitt, and A.D Salman. Drag correlations for particles of regular shape. *Advanced Powder Technology*, 16(4):363 – 372, 2005. ISSN 0921-8831. doi: https://doi.org/10.1163/1568552054194221. URL http://www.sciencedirect.com/science/article/pii/S0921883108608110.

[47] Wenqi Zhong, Aibing Yu, Xuejiao Liu, Zhenbo Tong, and Hao Zhang. Dem/cfd-dem modelling of non-spherical particulate systems: Theoretical developments and applications. *Powder Technology*, 302:108 – 152, 2016. ISSN 0032-5910. doi: https://doi.org/10.1016/j.powtec.2016.07.010. URL http://www.sciencedirect.com/science/article/pii/S0032591016304065.

[48] Z. Y. ZHOU, S. B. KUANG, K. W. CHU, and A. B. YU. Discrete particle simulation of particle–fluid flow: model formulations and their applicability. *Journal of Fluid Mechanics*, 661:482–510, 2010. doi: 10.1017/S002211201000306X.

[49] Zongyan Zhou, Ruiping Zou, David Pinson, and Aibing Yu. Discrete modelling of the packing of ellipsoidal particles. *AIP Conference Proceedings*, 1542(1):357–360, 2013. doi: 10.1063/1.4811941. URL https://aip.scitation.org/doi/abs/10.1063/1.4811941.