Efficient Numerical Methods for the Simulation of Particulate and Liquid-Solid Flows

Dissertation

zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
(Dr. rer. nat.)

Der Fakultät für Mathematik
der TU Dortmund vorgelegt von

Raphael Münster

im Februar 2016

Dissertation

Efficient Numerical Methods for the Simulation of Particulate and Liquid-Solid Flows

Fakultät für Mathematik
Technische Universität Dortmund

Erstgutachter : Prof. Dr. S. Turek
Zweitgutachter : Prof. Dr. H. Müller

Tag der mündlichen Prüfung: 21.07.2016

Contents

List of Tables

Chapter 1

Introduction
Efficient Numerical Methods for the Simulation of Particulate and Liquid-Solid Flows

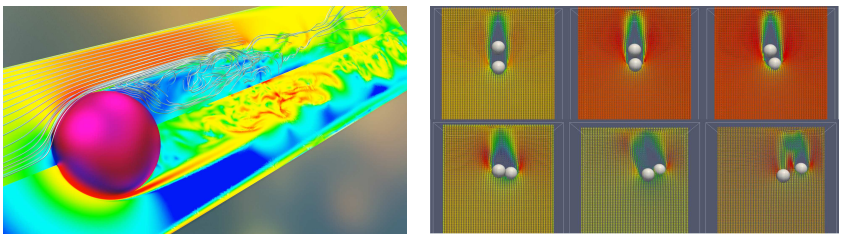
---

1.1 Introduction

This work is concerned with the numerical simulation of *particulate flows* and *liquid-solid* flow problems with complex geometries. Particulate flows are a subtopic of the larger field of *computational fluid dynamics* (CFD), which is concerned with the numerical solution of certain fluid equations on computer hardware. Some well-known test configurations in the CFD community are the calculation of the flow around a cylinder and, as an extension of this, the flow around an airfoil. The scientific methodology in CFD applications can in general be summarized as follows:

- **Theoretic Modelling** : Describes the problem to be simulated as a set of equations.
- **Discretization** : The simulation domain is discretized by covering it with a computational mesh. The mathematical equations can then be evaluated on computer hardware in the degrees of freedom of the computational mesh.
- **Numerical Solution** : The system of equations is solved by a numerical solution scheme.
- **Postprocessing** : The solution values are processed by a visualization software that helps the user to access the data intuitively and analyze it.

As we have mentioned the simulation of particulate flows is a subfield of CFD, some classic examples of particulate flow applications include the simulation of single particle sedimentation trying to compute accurate results for the particle's terminal free fall velocity or to simulate well-known particle interaction phenomena like drafting, kissing and tumbling (see figure 1.1 for graphical examples). Before we turn to the specific, detailed methods proposed and used in this work, we want to introduce and discuss the most important vocabulary and real-life applications in this field of research in order to provide an intuitive introduction to the topic. Particulate flows which alternatively can also be called particle-laden flows are characterized by a number of solid, rigid particles which are immersed into a carrier fluid. Thus, they are a special class of multi-phase flow problems that consist of two phases, the dispersed phase (the particles) and the liquid as carrier phase. When we talk about particulate flows we, in general, think of the particle phase consisting of particles with a simple geometry like spheres or ellipsoids. In case the geometry of the solids immersed into the fluid is



(a) Flowfield visualization around a simple particle

(b) Drafting, kissing and tumbling timeline from left to right, top to bottom

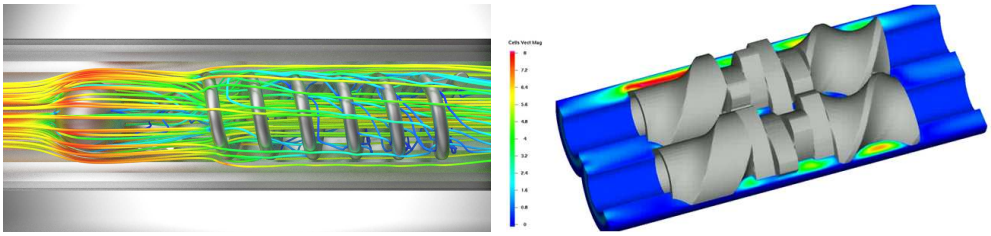
Figure 1.1: Prototypical examples of particulate flows

more complex, we talk about the more general class of liquid-solid flows. Typically, more complex geometries are considered when the flow around technical components or machinery used in various engineering applications needs to be simulated. When we try to assign a specific scientific category to the simulation of particulate and liquid-solid flows, we realize that multiple scientific disciplines are involved in this field of research, i.e. numerical mathematics, computer science, engineering, physics or chemical engineering. This aspect makes the study and simulation of particulate flows an interesting, challenging topic that connects researchers as well as scientific and industrial projects from various scientific backgrounds.
It is not surprising that realistic prototypical particulate flow applications are as well based on the aforementioned scientific disciplines. Examples for typical particulate flow applications include :

- Fluidized-bed reactors
- Separation and recycling of industrial waste
- Processing of natural oils

- Abrasive spraying.

These prototypical applications are characterized by a fluid medium interacting with multiple immersed rigid particles. The particles themselves are typically thought of as not only rigid, but also of a simple, in many cases spherical, shape. Also the number of particles in such applications usually ranges from hundreds to several ten thousands. Particulate flow applications in the broader sense involve more complex geometries than simple spheres. These applications we call general liquid-solid flows. In these



(a) Streamline flowfield around a helix geometry

(b) Liquid-Solid simulation of a twin screw geometry

Figure 1.2: Examples of liquid-solid flows with complex geometries

cases the focus of the simulation is usually to determine some specific features that a geometry has on the flow around it. Another difference compared to particulate flow simulations is that the number rigid objects is limited to a few objects, often not more than one. This is because the computational resources are limited and we aim to resolve the complex geometry by the mesh as fine as possible so that the interaction of geometry and fluid can be captured as well as possible. These kinds of liquid-solid simulations have a place in industrial production and planning processes. Different kinds of geometrical designs can be tested in the virtual testing environment without having to actually construct several prototypes for experiments and unnecessary waste of resources can be avoided. Apart from testing the influence of a specific prototype geometry on the application, the virtual testing environment can be used in following steps of the development pipeline (see figure 1.3) to optimize the the most promising geometry design with regard to the needs of the application. Some typical examples of these kinds of liquid-solid and virtual prototyping applications, some of which we will encounter in the course of the work, are:

- Virtual wind tunnels
- Virtual stenting
- Aneurysm rupture prediction
- Virtual work piece design and optimization.

In many virtual prototyping applications the geometry of the prototype is a quantity of interest and closely related to the desired features of the prototype. It is therefore of high importance in the simulation to accurately capture even small details of the prototype geometry. Another highly desired feature is the ability to easily change the prototype geometry and to quickly adjust the simulation accordingly. A software package that is able to handle particulate flow and liquid-solid virtual prototyping applications usually consists of a CFD-solver for the fluid part of the problem and of a coupled solver for the particle or rigid body problem. In our work we use as CFD-solver the FEATFLOW [27] solver package. The extension and the efficient coupling of the FEATFLOW solver with software components that allow the simulation of a wide range of particulate flow problems with complex geometries and liquid-solid virtual prototyping applications that are able to resolve even small details of prototype geometry are the main focus of this work.

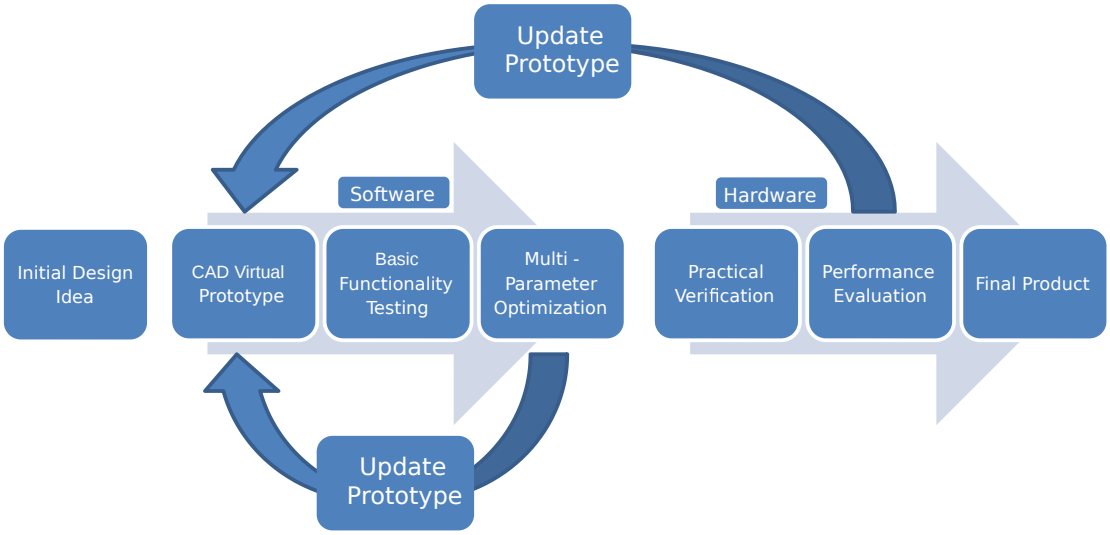


Figure 1.3: Workflow diagram for virtual prototyping

The idea of virtual prototyping is to aid engineers in product design (see figure 1.3). Without virtual prototyping the design of a product prototype can be a time and resource consuming process and one is reliant on the experience of the engineer to achieve the desired result with a minimal number of actual prototypes produced. With virtual prototyping availabe even without extensive experience the possibility to evaluate an initial design idea is provided by simulations. Recent research confirms the claim that the use of virtual prototyping can not only reduce the resources spent for the production processes, but can also improve the quality of the final result [36]. The virtual prototypes in simulations are usually represented by CAD (computed aided design) geometries, in the form of surface triangulations, the simulation framework then needs to process this geometry in the simulation and generate the computational mesh that discretizes the simulation domain. In the course of the simulation special tech-

niques need to be applied to increase the numeric efficiency of the simulation software. These can be the adaptation of the computational mesh to the shape of the geometry with these techniques the small scale details of the geometry can be captured and resolved in the simulation.
One of the main challenges in the proposed endevors is to develop a system that is able to provide accurate solutions to the problems and at the same time to satisfy demands with regards to compute time. The compute time required to provide competitively accurate results in thie earlier days of numerical flow simulation was not unusually seen to rise up to several months. With the help of more efficient theoretical methods and advances in compute hardware these compute times of the old days can be significantly reduced. The rise and availability of parallel hardware and compute clusters have led to the development of (massively) parallel CFD-software. These parallel CFD-solvers usually employ a domain decomposition approach that divides the simulation domain in several subdomains which in turn are then assigned to a CPU on the compute cluster. Following this development is the integration of special massively parallel hardware into these clusters like GPUs (graphics processing unit), multicore-CPUs and even programmable gaming console hardware. The advantage of these types of hardware is the ability to perform thousands of computations in parallel thanks to their high number of compute units. The potential acceleration of simulation codes is however not directly available. In order to gain performance by parallel hardware the algorithmic foundation of the software has to be such that it actually has the required data parallelism and independence to compute results simultaneously. It is the task of software developers to adjust their simulation codes accordingly.

1.2 Aims, Requirements and Scientific Context of this Work

The aim of this work is to design and couple a software module with the FEATFLOW software package that is able to provide the possibility to handle particulate flow simulations with simple spherical as well as complex shaped particles. The said additional software module should also be able to handle the virtual prototype applications where prototypes can be evaluated in a simulation and the geometric design of the prototype can be changed comfortably. Especially in the case of virtual prototype and liquid-solid applications with complex geometries methods that increase the numerical efficiency and accuracy by adapting the mesh to the detail of the geometry should be included. Furthermore, the evaluation of the methods implemented in this work is of great importance. As basis of this evaluations we will choose well-known benchmark test cases and experiments. For particulate flow applications the design of a rigid body simulation framework that is able to handle the interaction of multiple rigid bodies is central. The realistic practical applications that the particulate flow solver should be able to handle include the class of fluidized particle beds, particle sedimentation and

other typical examples that mainly originate from chemical engineering applications. The particulate flow solver should furthermore be tested in scenarios where there is in-

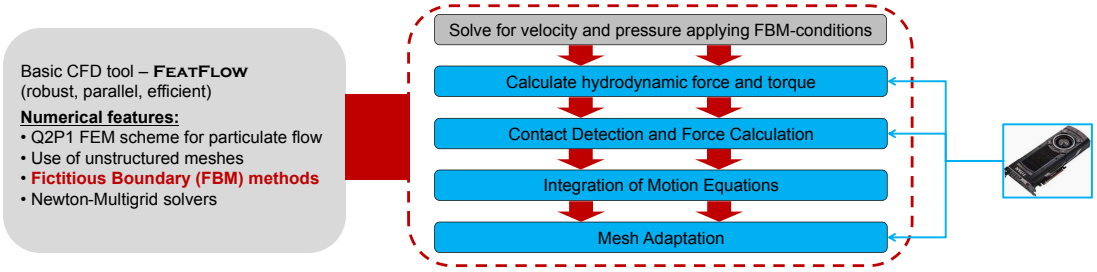


Figure 1.4: Overview of the FEATFLOW software package and the components that are part of this work. The topics marked in blue are the components that will be added or extended in the contact of this work. The arrows pointing from the GPU hardware indicate in which components hardware acceleration will be added.

teraction between multiple non-spherical particles in order to demonstrate and validate its capabilities in this still challenging class of applications. In the field of liquid-solid flows the general methodology how small scale details of the complex geometries are intended to be captured should be explained and analyzed. The validity and operability of the method should be tested in configurations with complex geometries (virtual wind tunnels, medical applications, etc.). With regard to the relation between theoretic approaches and their implementation the choice of methods in this work should always have in mind that the methods should be applicable and well suited for a parallel hardware implementation which is demanded by the domain decomposition approach that the parallel FEATFLOW solver uses. The interaction between the FEATFLOW solver, the additional components and extensions of existing components that will be dealt with in this work are depicted in figure 1.4. Additionally, the extension and partly the implementation of components of the software on massively parallel hardware like GPUs should be considered. That is, the choice of methods should be such that an implementation on GPU hardware is directly possible or several alternative approaches should be discussed that are suited to traditional CPU implementations and those that are more suited to GPU implementations.

1.3 Chapter Overiew and Structure of this Work

In this section we want to provide a short overview of the different chapters of this work and give a brief summary of each chapter. Our work is structured in a way that the different parts of it correspond logically to the components that are necessary to achieve the desired aims. The theory of the methods is described for each module of the final software package, whereas the validation test and simulation results are grouped in the final chapter.

In the first chapter we take a deeper look into the meshing aspect of our particulate flow solver. We briefly summarize basic theory and terminology of meshes and discuss the differences between basic mesh types that will be used in our work. The terminology is important to understand the acceleration structures used in several algorithms in the following chapters as well as for basic concepts of mesh adaption.

In our introduction to rigid body simulators we start with an overview of approaches to the design of such simulators. Special focus we will lay on the modular design of rigid body simulators. The chapter then continues to introduce the different modules of a modular rigid body simulator on a higher level, to explain their general functionality as well as their role and interaction with the software framework. After the general introduction of the modular components of the simulator we proceed to discuss the details and the theoretic concepts that are necessary to implement the different components in an efficient way. The components that will be discussed in detail are the broad phase module with its search data structures, the spatial temporal coherence analysis with its contact graph data structure and the narrow phase/contact generation module with its geometric algorithms to compute the contact information.

The following chapter focuses on a highly important module of the rigid body simulator which is the contact solver or contact force module. The module plays a central role in the simulation as it enables the simulation to handle multiple colliding particles which is a key feature in particulate flow simulations. The chapter begins with a short review of the necessary concepts from physics required to derive the mathematical formulation of contact forces. Based on these concepts and their relation to the contact force formulation a representation of rigid bodies in our simulations is proposed that is well suited to the context of our problem formulation. Different solution schemes for the contact forces are mentioned such as the constrained-based formulation, the sequential impulses formulation and a DEM-based approach which is well suited to an implementation on GPUs.

For the constrained-based approach to the contact force problem a special kind of mathematical problem arises: the linear complementarity problem. In a following section we briefly talk about the general theory of the linear complementarity problems (LCP) and formulate a solution strategy.

In the penultimate chapter we give a short summary of the FEATFLOW solver package and the algorithmic parts of it that we are going to use. The general flow equations are given and the basic solution methods is stated. Also the setup of the domain decomposition method and the partitioning into domains that are processed in parallel is touched. The fictitious boundary method (FBM) which is used as the interface tracking method and to couple the fluid equations with the equations of motion is introduced. Furthermore, we present efficient algorithms that are able to solve interface tracking

related problems in parallel as a new liquid-solid interface component of the FEATFLOW software package. We finish the chapter with a look at techniques that allow the adaptation of meshes to geometries that represent the solid objects in our liquid-solid simulations. We take a look at PDE and non-PDE based methods for adapting meshes to objects.

In the final chapter we want to evaluate and validate the methods mentioned in the previous chapters. The test cases presented include validation tests for a single particle to focus on the accuracy of the hydrodynamic force computation. Benchmarks for multiple particles and rigid bodies to test and compare the contact force solver to results of other research groups for the benchmark. As benchmark for the single particle configuration we have the well-known benchmark of ten Cate [13] that is concerned with the sedimentation of a single sphere towards a wall, our results will be compared to experiments as well as to the simulation results of other research groups. As multiple particle benchmark we have a fluidized bed test case by Aguilar et al.[1] where we compare the results for multiple particle based on statistic measures. Furthermore, we will be demonstrating the accuracy and the advantages of force computations with grid deformation with a simple oscillating cylinder benchmark and more complex realistic applications like the behavior of a microswimmer in a non-newtonian fluid at a low Reynold's number. The ability of the mesh deformation method to capture small scale details will be demonstrated in a virtual wind tunnel scenario.

Finally, we provide a list of literature references used in our work.

Chapter 2

Meshing
Terminology and Basic Theory

2.1 Introduction to Meshes

In this work we will encounter meshes in different contexts, one of them is to represent the surface of an object in a rigid body simulation or of a geometry around which we want to calculate the flow field. In a Finite Element simulation the domain and the immersed geometries are discretized by a so called *computational mesh*. The computational mesh covers the simulation domain with cells or *elements*. In 2D these elements are usually quadrilaterals and in 3D the usual choice of element type is the hexahedral element. That is formally surfaces are composed from polygons and volumes are constructed from Polyhedrons. The set of elements that cover the domain is in turn called the *mesh*. An alternative nomenclature for mesh is *cell complex*, but in this work we will continue to use the word mesh. For surface meshes that represent objects in a simulation triangles are the usual choice. We will now summarize the basic topological terminology and basic theory of meshes, the content in the following sections can be found in a more detailed and extended manner in the works of Kinsey [45] and Edelsbrunner[23]. As the first basic definition we will revisit the mesh:

> A mesh $\mathcal{M}$ is composed of a finite number of n-dimensional cells:
> $\mathcal{M} = \{\omega : \omega \text{ is a element}\}$.

The dimension of a mesh is equal to the highest cell dimension in the mesh. A cell of a mesh is the mesh component the inner of which is homeomorphic to the inner set of an n-dimensional disk. The boundary of a cell is composed of cells of lower dimension, these lower dimensional cell boundaries are called *faces*. For these relationship between faces and cells we can also use the notation $\sigma < \tau$ to indicate that

σ is a face of cell τ. A polyhedron for example is a cell that is composed of polygonal faces(two-dimensional cells), edges (one-dimensional cells) and vertices (zero-dimensional cells).

In order to construct a mesh we combine and connect cells of equal dimension: vertices are connected to vertices and edges are connected to adjacent edges. By $|\mathcal{M}|$ we refer to the set of points of mesh $\mathcal{M}$:

$$|\mathcal{M}| = \{\mathbf{x} : \mathbf{x} \in \sigma \in \mathcal{M}, \sigma \text{ is a cell in } \mathcal{M}.\}$$

$|\mathcal{M}|$ is also called the underlying space of $\mathcal{M}$. In order to define a topological structure for meshes, for every vertex in $|\mathcal{M}|$ a defined neighborhood relationship has to exist. In the example of a single cell mesh consisting of one polyhedral cell, we examine the neighborhood relationship. This mesh is composed of polygonal faces which in turn are composed of edges that are formed by connecting vertices. Here, we adopt the notion that vertices in the mesh correspond to the vertices in the lower dimensional cells that are used to construct the mesh.

For a vertex $\mathbf{v}$ that is located in the inside of a polygon, the neighborhood of $\mathbf{v}$ is an arbitrary disk that is entirely located in the inside of the polygon. If the vertex $\mathbf{v}$ is on an edge $\mathbf{e}$ that was contructed by unification of edges $\mathbf{e}_0 ... \mathbf{e}_n$ then the corresponding vertex $\mathbf{v}$ has to be present on every edge $\mathbf{e}_i$. The unification of half-disk neighborhoods of the vertices $\mathbf{v}_0 ... \mathbf{v}_n$ leads to the neighbor relationship that is depicted in figure 2.1. Furthermore, we see that in case that a vertex $\mathbf{v}$ is contructed from polygon vertices $\mathbf{v}_0, ..., \mathbf{v}_n$ the neighborhood of $\mathbf{v}$ is composed of the different disc- or half-disc neighborhoods of $\mathbf{v}_0, ..., \mathbf{v}_n$.

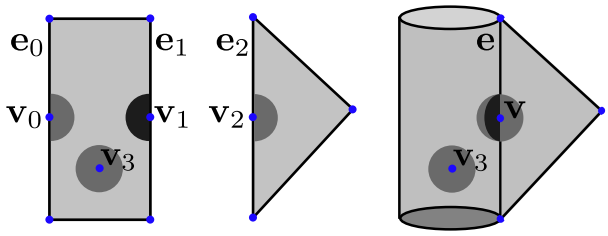
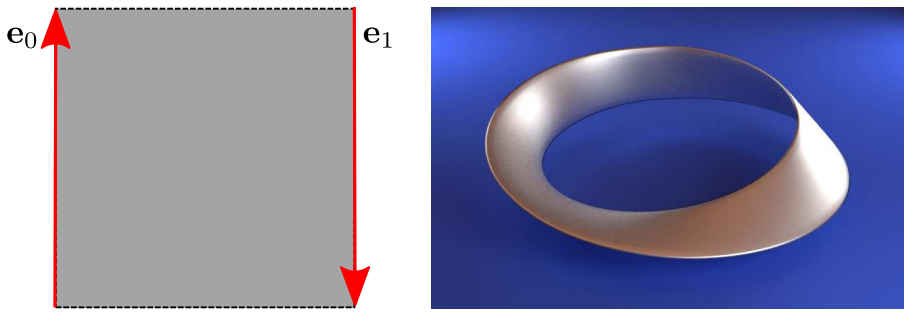


Figure 2.1: In the figure we unify the edges $\mathbf{e}_0$, $\mathbf{e}_1$ and $\mathbf{e}_2$, the resulting cell complex or mesh is on the right. The vertex $\mathbf{v}$ that has been produced by unifying the other vertices also has a neighborhood that is the result of the unification of $\mathbf{v}_0$, $\mathbf{v}_1$ and $\mathbf{v}_2$. Vertex $\mathbf{v}$ is an inner vertex and has a disk neighborhood that is fully inside before and after the edge unification [45].

2.2 Surfaces

The surfaces that we will use in the course of this work are *manifolds*, this is why we take a closer look at their properties. An n-dimensional manifold is a topological space in which every vertex has a neighborhood that is topologically equivalent to an open n-dimensional disk. The manifold definition also requires that two distinct points have disjoint neighborhoods. A 2-manifold is called a *surface*. If a manifold has a boundary then there exist vertices that have a neighborhood that is topologically equivalent to an open n-dimensional disk or half-disk. If a vertex has neither an open disk nor open half-disk neighborhood then it is called *non-manifold*. If a surface has one or more non-manifold vertices then the surface is called a non-manifold surface, see also figure 2.3 for examples of non-manifold surfaces.

A bounded manifold can be either *orientable* or *non-orientable*. Here the notion of being orientable or not is a global criterion, which cannot be defined by local features only. A surface is orientable if it does not contain a *Moebius Strip* [23]. A Moebius Strip can be explained by considering an $(n+1)$-dimensional object moving along an n-manifold. If the the object is located on one side of the n-manifold and on its travelling path it revisits a neighborhood, but this time from the other side, then this path is a Moebius Strip and the surface is not orientable. In a less abstract way a Moebius Strip can also be thought of as a strip of paper that is twisted and then glued together at an opposing pair of edges (see figure 2.2). The surfaces used to construct two-dimensional meshes are orientable, the cells of these meshes are triangles or quadrilaterals, which are connected by vertices or edges. Vertices or edges that are not part of a polygon should not be part of the mesh in the terminology that we have summarized in this section. From basic graph theory the adjacency and incidency relationships are important for our considerations. Adjacency is the neighborhood relation between same



(a) Planar representation of a Moebius Strip

(b) Moebius Strip visualization

Figure 2.2: A Moebius Strip can be created by twisting and connecting the edges of a planar strip, see also [23, 45].

type mesh components whereas incidency is the neighborhood relation between mesh

components of a different type. Using this terminology we call two polygons adjacent if there is an edge that is incident to both polygons. Using the incidency relationship we can define the notion of the stencil: the stencil of a vertex is the set of edges and polygons that are incident to the vertex:

$$Stencil\ \tau = \{\sigma \in K | \tau < \sigma\}$$

The degree or valency of a vertex $\mathbf{v}$ is the number of vertices $\mathbf{v}_i$ that is adjacent to $\mathbf{v}$, meaning there exists an edge that connects $\mathbf{v}$ to $\mathbf{v}_i$.

A manifold in the context of polygonal surfaces defines a configuration where for every edge a maximum of two surfaces are incident to that edge and for every vertex of the manifold only a single ring of surfaces is connected to the vertex. Furthermore, the Euler-Poincare equation

$$V - E + F = 2 - 2G$$

where V is the number of vertices, E the number of edges, F the number of faces and G the number of holes has to be satisfied for manifold surfaces. Cell complexes that do not satisfy the beforementioned manifold criteria are shown in figure 2.3.

Having discussed the topology and connectivity of polygonal surfaces in two di-

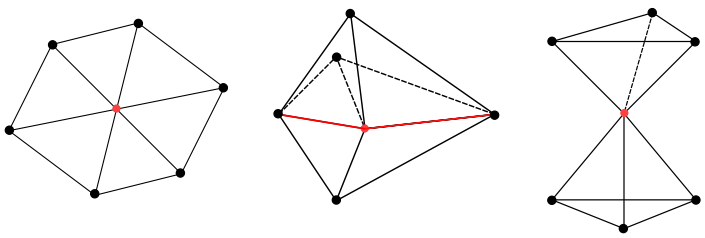


Figure 2.3: From left to right: manifold surface, non-manifold at an edge and non-manifold at a vertex.

mensions, we can begin to extend our considerations to the three-dimensional case. The concept of orientable manifolds remains the same for the extension to three dimensions and these orientable manifolds will be used in three-dimensional meshes for discretization by a finite element mesh in order to solve flow equations and as second purpose to represent three dimensional objects. These objects can be represented by a 3D surface mesh that describes the boundary surface of the object. The terminology that we have established before for two-dimensional meshes can naturally be extended to meshes consisting of polyhedral cells such as tetrahedrons, pyramids, prisms and hexahedrons (see figure 2.4).

2.3 Computational Meshes

In the preceeding section we have examined and introduced meshes from the point of view of topology. This work however is related to the numerical solution of flow equations on a mesh. Therefore a physical domain is discretized by our computational mesh in order to provide a finite set of elements where the related equations can be solved by an appropriate numerical solution technique. The process of generating meshes that are well suited for using them in a numerical simulation is called *mesh generation* and can be called a scientific discipline on its own right. The use of meshes is necessitated by the fact that the partial differential equations, that describe the physical phenomena that we want to simulate can be solved analytically only for very basic cases on simple domains. In order to solve realistic problems the domain needs to be discretized by a computational mesh so that the system of equations can be discretized. Some of the most popular mesh-based numerical solution techniques for PDEs include the Finite Volume method, the Finite Difference method and the Finite-Element method. The reason why we have briefly summarized the very basic foundation of meshes and their topology is that it allows us to formally introduce structured and unstructured meshes. In the remainder of this introduction to computational meshes we will examine structured and unstructured meshes and discuss their properties. In a numerical simulation the choice of mesh type and element type is dependent on the numerical simulation scheme and the concrete problem that needs to be solved. Not only the type of mesh employed is problem-dependent, but also the generation of the topological mesh structure depends on the type of physical problem and its specific features, the compute hardware that is available, the desired accuracy of the result, the available compute time and findings from previous simulations with different meshes. We can conclude from this list of dependencies that mesh generation is a highly complex topic involving requirements from various backgrounds.

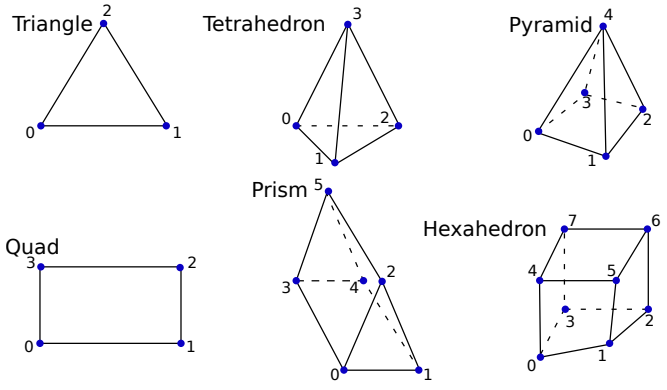


Figure 2.4: Typical mesh elements used in computational meshes.

2.3.1 Structured Mesh

A structured mesh is characterized by the feature that the degree vertices, the number of vertices adjacent to a mesh vertex, is the same throughout the whole mesh or in other words that it has a regular connectivity (with the exception of boundary vertices). The storage of such a mesh in computer memory can be realized by a simple array of cells. This most simple type of storage is discouraged in case the mesh is to refined in the course of the application, because new vertices need to be inserted which makes additional storage of connectivity information neccessary. Usual choices of elements in a structured mesh include quadrilateral elements in 2D and hexahedral elements in 3D. Elements like tetrahedrons are in general not used in structured meshes. The reason for this is that tetrahedrons require more elements to fill a domain with elements. The advantages of tetrahedrons, their ability to represent complex geometries better, do not play a significant role in structured meshes. As the approach in structured meshes to approximate geometries better is by additional refinement levels while keeping the mesh structure simple which is easier to do with regular hexahedral elements.

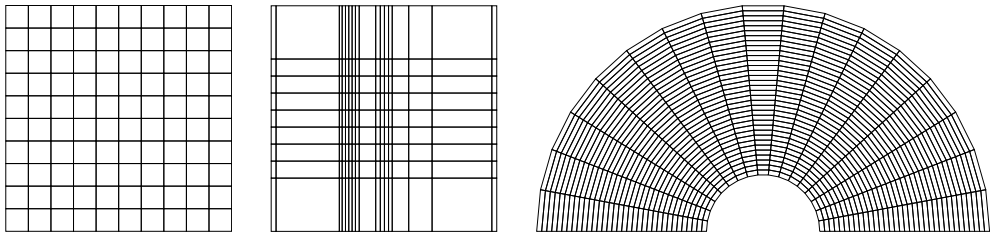
There exist different subtypes of structured meshes that are used as computational meshes which mainly include equidistant cartesian meshes, rectilinear and curvilinear meshes these are illustrated in figure 2.5. The equidistant cartesian meshes have many advantageous properties: they have a uniform element size and it is possible to easily locate an arbitrary point in the mesh, that is a relation between a location in space and an element can be established easily. For rectilinear and curvilinear meshes establishing this relation is also possible, but slightly more difficult. In terms of memory representation of such meshes this means that an element index in a data structure can be efficiently computed based on coordinates in space:

$$\mathbf{p} \mapsto i \text{ (array index)} , \mathbf{p} \in \mathbb{R}^3. \tag{2.1}$$

Equidistant cartesian meshes are often used for parallel finite volume, finite difference or general high performance computing schemes because of their easy generation, memory access patterns and simple domain decomposition capabilities. Besides their use in these applications equidistant structured meshes are used as acceleration structures for geometric computations where there are employed as a means to subdivide space. These properties make equidistant cartesian meshes and variants popular choices for spatial subdivision structures intended for acceleration of geometric calculations or other application specific geometric queries (visibility determination, point containment, distance or intersection queries).

2.3.2 Unstructured Mesh

In an unstructured mesh the vertices in general do not have a predefined uniform vertex degree. In principle this means that an arbitrary number of cells can be incident to a vertex. The freedom of not being confined to a uniform vertex valency allows



(a) Equidistant cartesian mesh (b) Rectilinear mesh (c) Curvilinear mesh

Figure 2.5: Structured meshes used as computational meshes, from left to right: equidistant structured mesh, rectilinear mesh and curvilinear mesh.

for better geometry resolution by the mesh, which is especially helpful in the case of complex geometries. Furthermore, unstructured meshes can employ connector elements to gradually reduce the level of refinement in a grid from domain areas where a high mesh resolution is required to domain areas where such a high mesh resolution is not needed (see figure 2.6). This property is a clear advantage over structured meshes where the mesh resolution can only be increased globally or by local vertex redistribution without changing the connectivity of the mesh. The choice of unstructured meshes in order to better approximate complex-shaped geometries introduces some difficulties as well. Complex geometries are often explicitly defined by a set of coordinates in space or by an explicit or implicit analytic function. Using an unstructured mesh there is in general no easy way to introduce a mapping from positions in space to element indices in computer memory. Which means that calculations that involve geometric information about the object that is represented by the mesh may need to resort to exhaustive search procedures if no measure are taken to add this feature to the class of unstructured meshes. Possible solutions to this problem are discussed in the following chapters. Standard choices for cell geometries in unstructured meshes are hexahedrons, tetrahedrons or prisms.

In comparision to structured meshes more memory is needed to store an unstructured grid for a numerical simulation. The additional memory is required because the connectivity is not defined implicitly anymore and thus neighborhood information needs to be stored explicitly.

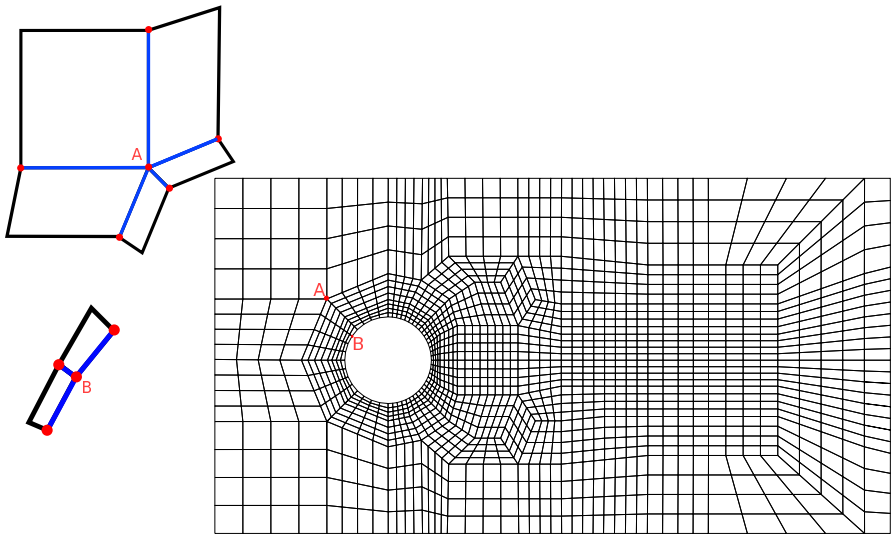


Figure 2.6: Unstructured 2D mesh where the mesh edges are aligned with an inner circle. We have marked some vertices with different degrees to that are used as anchor for connector elements (A) or as elements that approximate the circle geometry with its edges (B).

Chapter 3

Rigid Body Simulators
Design and Algorithms

3.1 Introduction

Rigid body dynamics are concerned with simulating the physical behavior of multiple rigid bodies and their interaction. A rigid body simulator is a generalization of a particle physics simulator as particles are seen as point masses with a simple geometrical shape. A true rigid body can then be distinguished from a particle as they appear in ideally arbitrary shapes and can have different material properties. Rigid bodies themselves are an idealization of general deformable bodies. However, the study of deformable bodies is not part of this work and we regard particles and rigid bodies as the building blocks for the theoretical methods used in constructing our rigid body framework.
As we have mentioned before numerical simulations are more and more becoming an integral part in many professions as well as development and construction processes. A quest for larger, faster, more complex and more realistic simulations is clearly visible. Such complex simulations usually involve not only one simulator or only one simulation paradigm, but they couple multiple different simulators like CFD-simulators, rigid body simulators, chemical simulations, laser optics simulations or physically-based rendering. These different simulators can then be called the individual modules of a more complex physical simulation, here the different simulator modules may themselves again exhibit a modular design. This is especially true for our rigid body framework, adopting a modular design enables us to easily use different methods for the individual parts of our simulator such as collision detection strategies, collision force solvers or time-stepping schemes. Additionally, the modular approach helps to present the theoretical methods of the different parts of the simulator in a structured

manner.

We start off with an overview of rigid body simulators and modular design possibilities, afterwards we will turn to the collision detection module (called the broad phase) which heavily focusses on different grid structures that allow us to accelerate distance and proximity calculations. The grid data structures presented for collision detection are furthermore used in other areas of our framework as acceleration data structures for general geometric problems. The collision detection module consists of the broad phase, a middle phase and the narrow phase. The following component that will be discussed is the middle phase. The middle phase is an intermediate step in the simulator that over time collects relational information between the rigid bodies like collision states as well as cached information of the collision states to speed up computation and other quantities that evolve of the time of the simulation. The idea of the middle phase has in recent rigid body simulator designs been extended [37, 25] to be active not only as the name middle phase suggests between the broad and the narrow phase, but also to be active after the narrow phase and after the calculation of contact forces. This extended idea of a module that tracks relational information about rigid bodies has been associated with the term spatial-temporal analysis module [25]. As next module of the simulator we will deal with the narrow phase. The narrow phase is the part of the collision detection module that computes all the neccessary input for the collision response solver. The information computed in the narrow phase is mainly of geometrical nature such as the point of contact between two rigid bodies, contact normals and similar quantities. The outline of the rigid body simulator we are going to contruct is shown in figure 3.1.

The topic of calculating a collision response when our detection module has reported a collision will be dealt with in chapter 4 because it is a larger complex module.

The integration of the equations of motion will be dealt with in the collision response chapter as this step is tightly coupled with the collision response. The discussion of the implication of the local time-stepping of the rigid body simulator in the case when it is an embedded module as part of a particulate flow simulation will also be delayed until the collision response chapter as the dependency between those two topics is very strong.

The goal we want to pursue is to build a rigid body simulator that is able to be integrated in different simulation frameworks that want to extend their original simulator with rigid body simulation. In order to provide such an interface to a rigid body simulator an efficient approach is to only exchange the relevant physical quantities that are needed for either side of the simulation. Our approach to accomplish this is the addition of a module called the external force module which will act as the interface between the CFD-solver and the rigid body simulator, the detailled realization of this module will be explained in chapter 5.

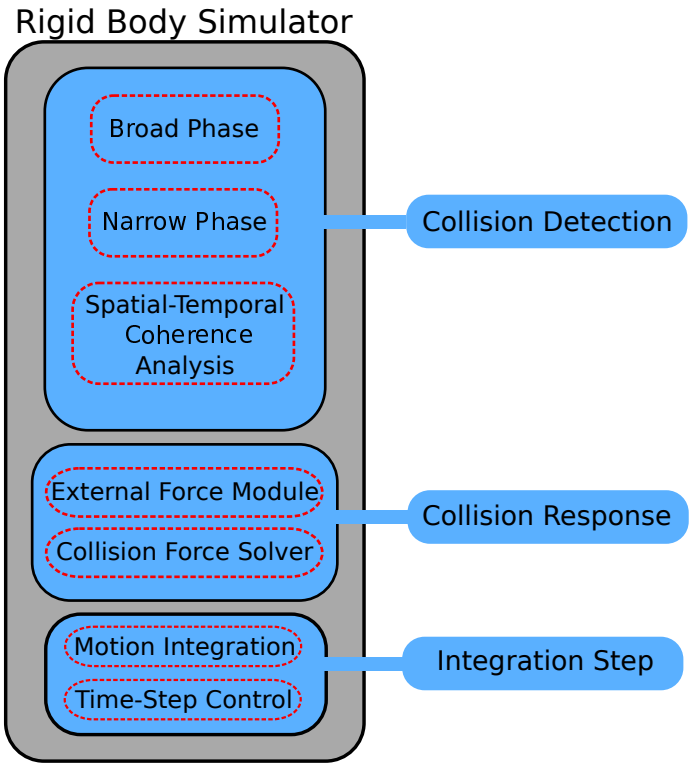


Figure 3.1: Rigid Body Simulator Outline

3.1.1 Overview of Rigid Body Simulator Design

In this introductory section we want to discuss design considerations for rigid body simulators from a high-level abstraction point of view, in other words we are not focussed on details, but general aspects. We said, a rigid body simulator consists of several modules, a module is a component that fulfills a specific task in the simulator, each module may be broken down into submodules. On the highest level the structure of a rigid body simulator is shown in figure 3.2 which is the standard since the introduction of modern day rigid body simulation [56, 6]. We see that in its basic form a rigid body

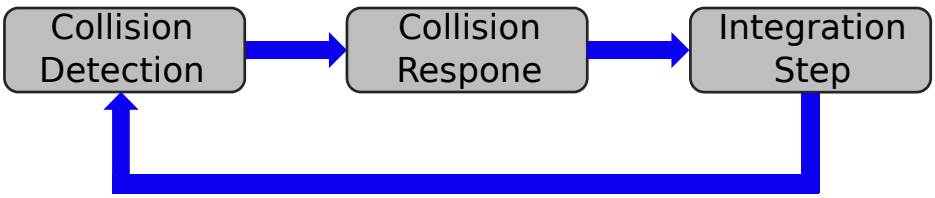


Figure 3.2: Simplified simulation loop of a rigid body simulator.

simulator consists of a collision detection module, a collision response module and a module that handles the integration of motion equations for the rigid bodies and thus results in an updated position of the objects in the simulation. This is the modular rigid body simulator setup that was proposed by Erleben [25]. The advantage of this setup

is that different methods and algorithms can be 'plugged' in as modules to customize the simulator for the specific simulation task that should be performed. For example the collision response module can be realized by very different approaches like:

1. Constrained-based methods[5, 6]
2. Penalty-based methods [83, 21, 99, 98, 65, 44]
3. Impulse- or Sequential Impulse-based methods [37, 56]
4. DEM-based methods[8, 66, 57, 39]
5. Hybrids and combinations of the above methods [53].

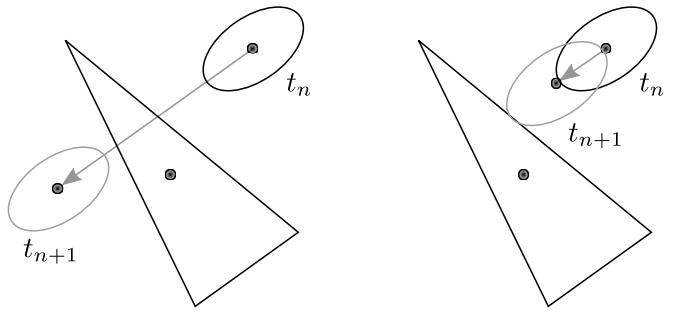
The collision detection module from figure 3.2 is a purely geometry-related component, the modules collision response and integration are related to the physics simulation. Therefore it is possible to group them in a larger simulation. We will briefly give a general overview of these modules to illustrate their basic functionality, interaction and discuss design decisions before explaining the detailed theory and methods used. We start by looking at the submodules *Timestep-Control* and *Motion Integration* of the Integration module as shown in figure 3.1.

Timestep-Control Module

The timestep-control module controls the start and length of the simulation. The module closely interacts with the motion integration and the collision response solver. The timestep size is chosen according to the fixed or adaptive time-stepping policy. One possibility to implement the adaptive time-stepping strategy in rigid body simulation is the backtracking approach [3, 55]. The idea of this method is to step forward in time by a user-defined timestep Δt until the collision detection module reports a collision. Usually when the we step forward like this the reported collision will involve interpenetrating rigid bodies which is an undesired state. When an interpenetrating collision is reported the simulation will then rewind back to the last known point in time that was free of penetrating rigid bodies which is why this scheme is also called retroactive detection (RD) [74]. The simulation then steps forward in time with a smaller time step and the procedure will be repeated until the exact point of touching, non-interpenetrating contact is found and can then be handled by the collision response solver. This process is similar to numerical root finding algorithms. Another type of adaptive time-stepping scheme is based on continous collision detection (CCD) [69, 70].
In CCD an estimate for the earliest time of impact (TOI) between the candidates for collision that have been determined by the broad phase is calculated. These time of impact values are then stored in a heap data structure. The time-stepping algorithm then chooses the timestep Δt as the difference between the current time and the TOI estimate of the collision pair on top of the heap data structure. The simulation will then

step forward in time and update the TOI estimates in the heap. It is important to note that in this scheme the TOI estimate is conservative meaning that it will usually calculate an estimated TOI that is smaller than the actual time of impact between the rigid bodies. This way it is guranteed that an interpenetrating configuration does not occur. Fixed time-stepping always step forward by a constant time step Δt. Therefore they are the most simple and least computationally expensive schemes, but these methods have their drawbacks. Fixed time-stepping schemes usually result in interpenetrating rigid bodies or even in the case of very large time steps in missed collisions, an effect which is also called tunneling (see figure 3.3). When a fixed time-stepping scheme is used it is absolutely neccessary to use a sufficiently small time step so that deep penetrations do not occur and use a robust collision response solver that can deal with small interpenetrations.

In our situation where the rigid body simulator is embedded into the CFD-simulation



(a) Collision missed by the system due to large time step size

(b) Collision can be detected and reported with a smaller time step

Figure 3.3: The tunneling effect

the time stepping of the CFD takes precedence over the rigid body simulator. In almost all cases the time step of the CFD-solver for non-stationary simulations will be small enough or even smaller than the value that is required for the rigid body simulator to use a fixed time-stepping scheme and still avoid deep interpenetrations and unphysical states. In this case no significant benefits are achieved by using other schemes than the fixed time-stepping scheme. In case the CFD-solver uses an adaptive time-stepping scheme we can still observe the phenomenon that maximal timestep size of the CFD-solver in the adaptive scheme is more than adequate for the rigid body simulator, so again the rigid body simulator can step forward using the same timestep as the CFD-solver.

Motion Integration Module

The motion integration module and the timestep-control module are closely related. The timestep-control module takes the current timestep size Δt and passes this value to the motion integration module. The motion integration module itself handles the

position update of the rigid bodies in the simulation. The update is computed by a solver for ordinary differential equations (ODE) that solves the equations of motion. The execution of the motion integration module is usually the last step in the simulation loop of the rigid body simulator. In order for the motion integration module to function properly collisions have to be detected, appropriate collision forces and eventually other external forces have to be calculated. The motion integration module is also responsible for caching information about the previous positions and state of the physics world in case that a backtracking time scheme is used.

External Force Module

This module is responsible for the calculation of any external forces that might act on the rigid bodies in the simulation. In our special case of coupling a CFD-solver with a rigid body simulator these forces will, of course, be the hydrodynamic forces that are acting on the bodies interacting with the fluid. So an interface to the CFD-solver has to be created that can be queried for the forces acting on a body based on its current velocity, position, orientation and the neccessary physical properties like density, mass and volume. The result of the query will then be the hydrodynamic force $\mathbf{F_h}$ and the torque $\mathbf{T_h}$ acting on the rigid body. The functionality of the interface between the CFD-solver and the rigid body solver is shown graphically in figure 3.4. The internal realization of the force computation in the CFD-solver is an important topic. For the computation of the hydrodynamic forces the cells of the computational mesh of the CFD-solver are examined to see if they coincide with the interior of a rigid

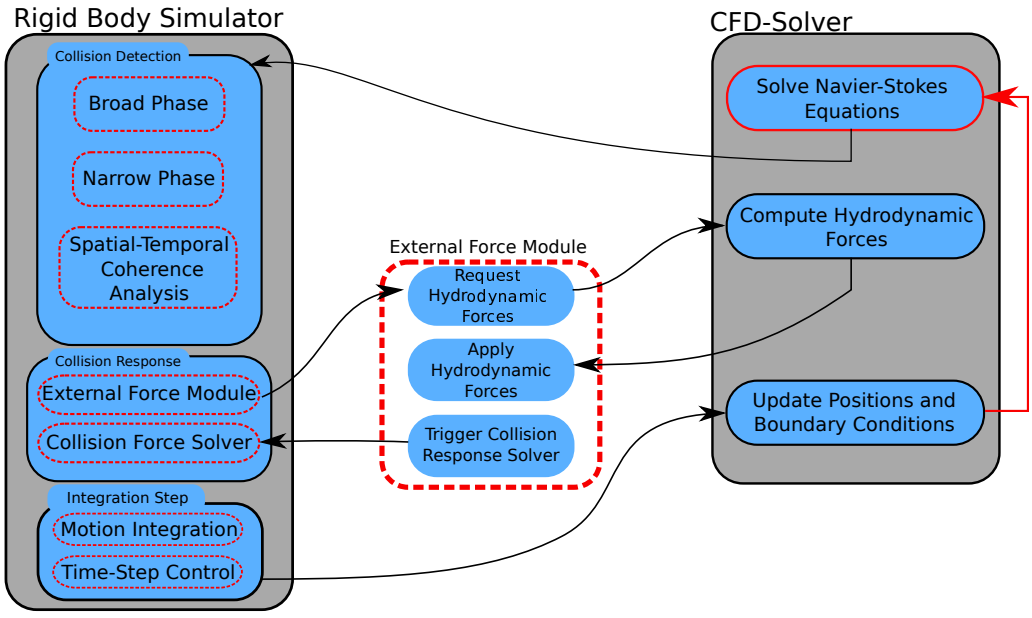


Figure 3.4: Rigid Body Simulator Outline

body, after having determined all the cells that constitute the interior of a rigid body

the hydrodynamic forces can be calculated. It is clear that a brute force approach to solve this problem results in an algorithm with a complexity of $\mathcal{O}(nm)$ where n is the number of cells in the computational mesh of the CFD-solver and m is the number of particles. So the external force modules techniques are needed to reduce the cost of this operation. Possible solutions are the overlaying of the compuational mesh of the CFD-solver with a spacial search grid structure which allows an operation that is not possible with the unstructured grids that are used by the CFD-solver. This operation is the determination of cell indices in the mesh data structure based on geometric coordinates as we have seen in eq. 2.1. The availibity of such an operator would eliminate the need to search the whole mesh and reduce the search to only a few cells around each rigid body. Further strategies to reduce the compute time of the hydrodynamic forces is the caching of the cells that are inside of the rigid body and then a fast update procedure can be invoked everytime the objects move. The acceleration techniques play a cruical role in the concrete implementation of the external force module as this step has the potential to slow down the simulation greatly if it is not efficiently implemented.

Collision Response Module

The collision response module is called when all force computations for the rigid bodies are finished. With all the force information available a total force acting on each rigid body can be computed which will be used in the collision response solver. The purpose of the collision response solver is to compute a physical reaction to the collision of rigid bodies and to impose the non-penetration constraint. The non-penetration constraint enforces that no pair of rigid bodies is occupying the same volume and thus displaying deep interpenetrations. Due to numerical inaccuracies it is nearly impossible to enforce absolutely no penetrations at all no matter how small it may be. It is advantageous to relax the non-penetration constraint and allow some small amount of penetration, otherwise a collision response solver will just fail to find a solution in some numerically critically situations or, in general, use an unneccessary amount of compute time in order to correct some neglegible penetration effects. A collision response is usually computed in form of an impulse. An impulse is a force that triggers an instantaneous (discontinuous) change in the motion of the rigid body. Let us imagine the simulation of a ball falling towards the ground: gravity is acting on the ball and accelerating the ball towards the ground. When the ball is making contact with the ground the collision detection module detects this and adds a signal for the collision response module to compute the forces for this collision. The collision response solver will then calculate an impulse that not only prevents the ball from penetrating the ground, but also enforces a reaction with respect to a chosen collision model. In this case the collision response might calculate an impulse for an elastic collision. The impulse is then applied to the ball and it instantaneouly changes its velocity. Impulses in general are calculated according to a certain contact model. There exist different classes of contact models that differ with regard to their accuracy of representing the actual physical behavior of colliding bodies and the computational cost that is required

to solve the contact model. The most accurate contact models consider a full deformation law which computes the compression and expansion of a rigid body during a collision based on partial differential equations (PDE). Other contact models are based on algebraic laws that lead to a system of equations which can then be efficiently solved. These equations are based on the known physical pre-collision states and desired post-collision states. We can then solve these equations for the post-collision states to get our impulses. Furthermore there are incremental collision models that build up the collision forces over a certain collision time which can then be integrated to get the accumulated collision force. To compute the forces by an incremental model is usually more computationally expensive than using an algebraic law, but these models do not require a solver for a system of equations. In our implementation we included incremental and algebraic models for collision forces. Introducing another PDE-solver for the rigid body collision besides the CFD-solver would be too expensive for the type of applications we plan to handle with our setup. When i.e. several thousands of collision problems have to be solved the percentage of the computation time for a single time step of the simulation that has to be spent in the collision response module would be too high. Impulses themselves can be applied to rigid bodies either simultaneously [37] or sequentially [57, 55]. Simultaneous application of impulses leads to a system of equations where the impulses for a specific collision pair i are affected by the impulses of collision pairs i_j that are connected to the pair i. This way impulses are computed and applied in a single time step. For sequential application of impulses the impulses are applied pairwise and the propagation of impulses takes place over several time steps. Usually, for sequential application of impulses a lower time step is required.

Collision Detection Module

The collision detection module call follows the motion integration. Once updated positions are computed the collision detection checks for new possible collisions. The purpose of the module is to provide in every time step a list of the pairs of rigid bodies that are in a colliding state and to calculate all neccessary geometrical information required by the collision response module. Different approaches to collision detection exist: the continuous collision detection (CCD) approach is based on the premise to prevent interpenetrating collisions by trying to determine the earliest time of impact of a pair of rigid bodies and then step forward in time with an appropriate time step to the time of impact and trigger the collision response solver. The other approach is to allow for small interpenetrations, use a contact tolerance (based on distance) for that two bodies are considered in a colliding state, employ a linearized contact condition [73] which can be regarded as a heuristic to determine a colliding state or use hybrids of these criterions. The collision detection module has clearly defined submodules:

1. Broad Phase Collision Detection
2. (Middle Phase Collision Detection)
3. Narrow Phase Collision Detection

4. Contact Point Determination

5. Spatial-Temporal Coherence Analysis.

The Spatial-Temporal Coherence Analysis module was introduced by Erleben [25] and partly by Guendelman [37] as they share the same data structure, the *contact graph*. It can be said that this module contains the full functionality of what some call the middle phase where bookkeeping about the relation of pairs of rigid bodies is done and their state is tracked over several time steps, also quantities that develop over time are tracked, additionally Erleben extended and enriched this module with an analysis of collision pairs by means of a contact graph that adds valueable information that can be used in different ways to improve the simulation. The implication is that in a setup where a STC module is used the middle phase is obsolete. In the following sections we will briefly introduce the different components of the collision detection module.

Broad Phase Module

In the Collision Detection module the colliding pairs of rigid bodies are determined, here one problem immediately comes to mind: in the worst case the number of intersections checks that need to be performed is $\frac{n(n-1)}{2}$ where n is the total number of objects. While there is no way to reduce the worst case complexity, we can reduce the number of checks that needs to be done on average significantly. Because only one rigid body can occupy a volume at a certain time the worst case configuration where every object is in contact with all others is highly unlikely. In most situations a rigid body can possibly only collide with a limited number of rigid bodies in its spatial neighborhood. The task of the broad phase is to reduce the number of checks required severly by excluding those pairs of rigid bodies that cannot possibly collide, which is of great importance for the performance of the simulation, because for those pairs we do not need to perform the narrow phase collision tests which contain computationally expensive geometrical intersection tests. The typical approach to this problem is to use search or pruning data structures like uniform grids, hierarchical grids, tree-based structures or use sweep and prune strategies. We will discuss our choice of data structure in section 3.2 as well as other low-levels details of the broad phase implementation.

Narrow Phase Module

The narrow phase module takes as an input the list of potentially colliding pairs that have been determined by the broad phase. Let us recall that it is not the task of the broad phase to finally decide whether two rigid bodies are colliding, but to exclude those pairs of rigid bodies that cannot collide at the current time of the simulation from further processing. It remains to be the task of the narrow phase to finally and precisely answer the question whether the pairs of rigid bodies that come from the broad phase are really in colliding contact. The narrow phase module contains algorithms that can

determine intersections between numerous different analytical geometrical shapes or shapes that are represented as surface triangulations. These algorithms usually can additionally compute information that is important for the computation of collision forces like contact normals.

Contact Point Determination Module

The contact point determination module is closely tied to the narrow phase and often part of the contact point determination is already done in the narrow phase. In these cases the contact point determination module bundles and completes the neccessary information. The main task of this module is to compute the points of contact or the contact points between two colliding rigid bodies and the contact normal. The contact points are important because these are not only required in the computation of the collision forces, but are also the points where the collision force is applied to the rigid body. Additionally, the collision normal is required because it is needed to define the direction in which the contact force should act. If contact information over several time steps is available like the collision force in the last time step, contact, distance or penetration depth information then it is also added to this bundle, as it will help to simplify or accelerate the computation.

Spatial-Temporal Coherence Analysis Module

In a simulator according to the design of Guendelmann or Erleben after the motion integration step the STC analysis module is triggered. This module is a collection of methods that analyze different relations and collision states of the rigid bodies and quantities that develop over time in order to cache the results of the analysis for the next time step so they can be used to accelerate computations in the narrow phase, contact determinations and collision force solver. One of the central instruments that Guendelmann and Erleben introduced is the so called *contact graph*. A contact graph can help to identify the collision state of rigid bodies over time, to identify contact groups and to provide geometrical information that can help to speed up the collision response solver. Contact groups are independent connected components of rigid bodies, independent groups of rigid bodies have the advantage that they can in principle be handled in parallel. Furthermore, if the contact graph provides information on the spatial relation between rigid bodies this information can be used to calculate the propagation of collision forces more efficiently. The STC module is an optional module that is mainly introduced for efficiency reasons.

3.2 The Broad Phase

The broad phase is the module which is first called in the simulation loop of our rigid body simulator. The module is concerned with providing an efficient solution to the n-body collision problem that in all, but the most malign cases avoids the worst case

complexity of $\frac{n(n-1)}{2}$ collision checks. The way to severly reduce the number of collision checks in the broad phase is to use acceleration data structures that allow us to quickly generate proximity information between potentially colliding rigid bodies and to efficiently exclude those pairs of rigid bodies that cannot possibly collide in the current time step. The data structures that are constructed to aid these calculations partition the 3D space (and hence are often called *spacial partitioning structures*), this way we can infer that rigid bodies can only collide if they are located in the same or directly neighboring partition of the simulation space. In rigid body simulators the candidates for spatial partitioning data structures that are usually considered are grids, trees and spatial sweeping techniques [24]. In this section we will explore which choice of data structure is best suited to the needs of rigid body simulator that we wish to construct.

Uniform Grids

A popular choice of spatial partitioning structure is a uniform grid that covers the simulation domain with cells of a fixed size. Each cell gets assigned the objects of the simulation it contains. If the cell size is chosen properly with respect to the size of the rigid bodies, these grids allow us to only consider for collision tests only objects that are assigned to the same cell and to the neighboring cells which would be a total of 27 cells in 3D. The implementation of a simple uniform grid would lead to an array (representing each cell of the grid) of linked lists (used to store the rigid bodies). The advantages of uniform grids are their simplicity and the efficiency of mapping between coordinates in the simulation domain and elements of the grid. This most simple form of unifom grids however suffers from serious disadvantages. The first issue is related to the fixed cell size, when the size of the grid is fixed and the size of our rigid bodies is not, situations can arise where the grid is too fine (when the object is to large compared to the cell size), or the grid can be too coarse compared (objects too small for the cell size) or even too fine and too coarse at the same time when the size of the objects is significally different. We illustrate the different situations in figure 3.5. This first problem of simple uniform grids can be handled by using a hierarchy of uniform grids where objects of different sizes are assigned to different grid levels which use different cell sizes. All that we need to do is to overlap the cells not only with the neighbors, but also with the cells on the other levels in the hierarchy to find potential collisions. Another problem is that potentially a simulation domain can be very large and objects like particles can be small in comparison to the domain size, this way a grid with a sufficiently small cell size is needed leading to a large number of cells and hence more memory usage. Since we are thinking in terms of a rigid body simulator coupled with a CFD-solver we need to conserve resources as also the CFD-component manages a sophisticated grid structure that potentially can consume a lot of memory. We also note that often a lot of the cells of the uniform grid may not be containing any objects when the rigid bodies in simulation gather in a specific part of the domain. For these issues the idea of a hashed grid storage offers an elegant solution.

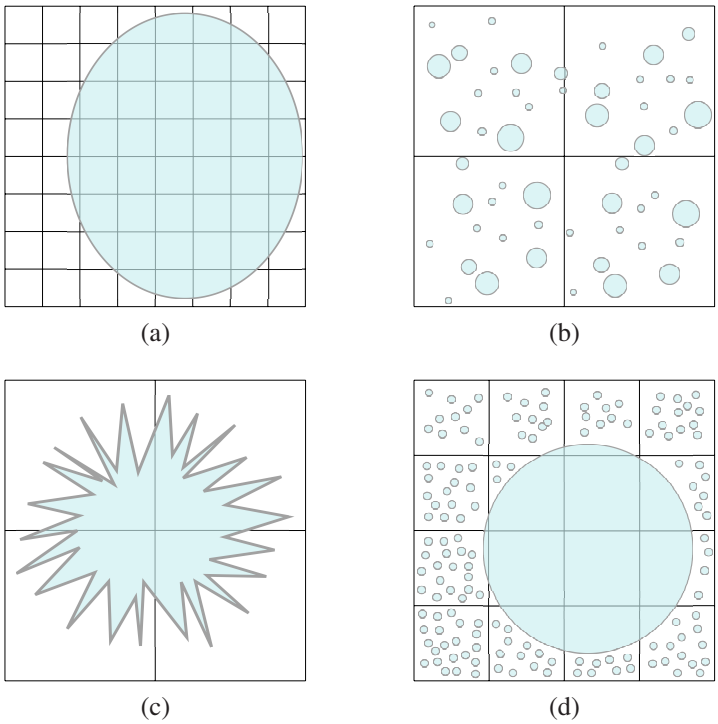


Figure 3.5: In case: (a) the grid is too fine, (b) the grid is too coarse, (c) grid too coarse with regard to geometric complexity, (d) the grid is too coarse and too fine, see also [24].

Hash Grids

The typical implementation of a uniform grid is an array of the cells of the grid with each entry of the array being a linked list that contains the objects that are associated with this cell. A hash grid on the other hand maps each cell of the grid to a set of n so called buckets by a hash function h:

$$h : (x, y, z) \mapsto \{0...(n-1)\}. \tag{3.1}$$

We have thus created a grid that can potentially contain an infinity number of cells, this is why hash grids are also called infinite grids. Thus infinite grids have the convenient property that they do have an open boundary, so any special case handling for boundary cases is not neccessary. The mapping hash function determines how likely hash collisions can occur, for well-suited sophisticated hash functions we refer to [61]. A hash collision also means that more collision tests than neccessary are performed when hash collisions are not treated, because we will have objects that are in the same bucket, but not in the same cell that we wanted to check. If however hash collisions occur they can be handled by either closed hashing or open hashing [61, 24]. In open hashing the bucket itself contains a linked list of entries for each of the cells in the bucket whereas in closed hashing the bucket contains directly the entires themselves.

If the entries of a bucket contain an identifier for the cell they are supposed to be in. Then the closed hashing strategy can be used and all relevant operations like deletion, update, insertion can be performed directly without any additional overhead. Also no unneccessary collision tests need to be performed because those objects in the bucket with different cell identifiers can directly be skipped. To avoid problems arising from different object sizes the hash grid structure can be organized hierachically.

Alternatives: Sweep and Prune, Trees

Sweep techniques usually perform sweeps of the simulation domain in the direction of the world coordiante axes checking for possible bounding box interval operlap of the objects. The tests for bounding box overlaps are performed at discrete locations in space using fixed steps. Sweep and prune algorithms do not maintain a grid data structure like the other approaches we have discussed [24].

Discussion

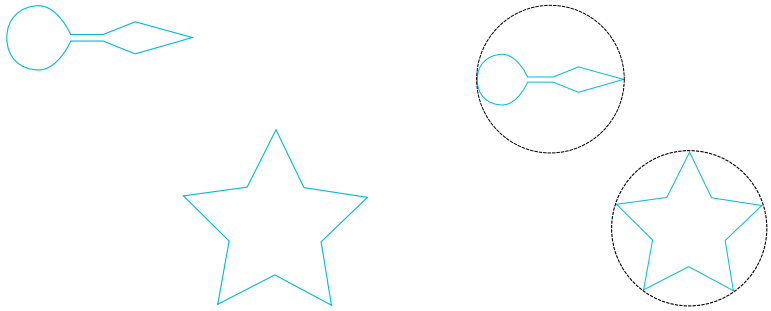
Sweep line techniques have the advantage of not needing additional memory to store i.e. a grid and they also offer a good speedup. The argument against the use of sweeping techniques are that the methods that use a search grid data structure have more to offer. The search grid can be used for multiple purposes in the simulation like providing the ability to map coordinates in the simulation domain to grid cells, to accelerate distance and point classification computations. So the cost of using additional memory is balanced by the additional functionality and performance in other components of the simulation that is gained by using it.

Collision Testing with Hierarchical Hash Grids

We will now examine in detail how collision tests can be efficiently performed with hierarchical uniform hash grids. When using a grid for collision testing the following question have to be answered:

- How should the grid cell size be chosen?
- How should the objects be assigned to the cells of the grid?
- How many grid levels are needed?
- Should a fixed number of grid levels be used or could grid levels be added on the fly?
- How to efficiently test for collisions?
- How update the search structure when the objects change position?

The first step in building the grid structure is determining how many grid levels are needed. This is directly dependent on the number of different object sizes that are present in the simulation. In order to determine the number of grid levels needed different heuristics exist. We can for example use as many grid levels so that on any given level we do not have objects in a cell whose bounding volumes differ by more than a certain percentage λ where we empirically see good results for $\lambda \approx 0.25$. When building the grid structure it is generally advantageous to look at the bounding volumes of the objects rather than the real geometry with all possible details as we are interested only in quickly pruning away objects that cannot possibly collide. Another question when building the grid is how exactely the object cell relation should be established. We already explained that in the broad phase we look only at the bounding volumes of the objects. Let us assume we are using spherical bounding volumes for the objects (see figure 3.6). As we can see in figure 3.6 a spherical bounding volume may be a



(a) Two prototypical rigid bodies

(b) Spherical bounding volumes for our rigid bodies

Figure 3.6: Objects and their bounding volumes

better geometrical approximation for some objects and for other objects they may be very different from the actual geometry of the object, but for a broad phase collision check this is still sufficient. Also the use of bounding spheres has the advantage that it is a bounding volume that is not affected by the rotation of the object as i.e. would be the case with axis-aligned bounding boxes. We could then assign objects to cells with respect to the center of the bounding sphere, but: should we also store the objects in the cells that are overlapped by the bounding sphere? Storing the objects only in the cell where the bounding sphere center is located would result in a lower number of entries per grid cell on average and it would require for each cell to check the neighboring cells to find all potential collisions. Moreover, if we proceeded this way the collision detection system would detect collisions between one and the same pair of objects multiple times, which is an undesireable effect from this method and we would have to add measures to filter out those multiple reported collision pairs. We will show how we have to set up our grid so that we can store an object only in the cell where the center of its bounding volume is located and how we can be sure that it is sufficient to check only the 26 neighbors of the cell to find all potential collisions. For each level l of the

grid we set the cell size to be h_l, we then make sure that for every object o_{l_i} on level l with bounding sphere $bs(o_{l_i})$ and bounding sphere radius $rad(bs(o_{l_i}))$ the equation:

$$h_l > 2 \cdot rad(bs(o_{l_i})),\ \forall l, \forall i \in \{0...n_l - 1\}, \tag{3.2}$$

where n_l is the number of objects on level l, is fulfilled. This way we can be sure that any pair of objects on any given level cannot collide if they are separated by more than one cell (see figure 3.7).

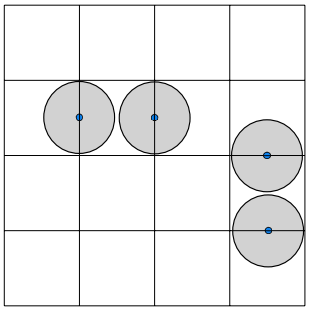


Figure 3.7: Pruning method: objects that are separated more than one cell cannot collide.

With a grid set up like we discussed before the procedure to find all potential collisions using our grid can be formulated as follows:

Algorithm 1: Basic procedure for finding potential collisions in a uniform grid

```
Data: grid G
Result: list<CollisionPair> broadPhaseCollisions
begin
    foreach cell c in grid G do
        if !c.empty() then
            foreach neighbor n of c do
                foreach potential collision pair p do
                    broadPhaseCollisions.pushback(p)
```

A problem with the basic algorithm 1 is that it also reports some collision pairs multiple times because it always checks all neighbors for each cell. Another possibility would be to traverse the objects of the simulation instead the cells of the grid, find for each object the cell it contains and check each neighbor of this cell. This would prevent us from traversing potentially emtpy grid cells, but still not solve the problem of detecting collisions multiple times. To solve this problem we can traverse the grid in a structured manner moving from one cell to a directly adjacent cell so that we can skip checking

basically half of the neighboring cells because we have already performed the tests when we were processing the last cell. We illustrate this way of checking the grid in figure 3.8.

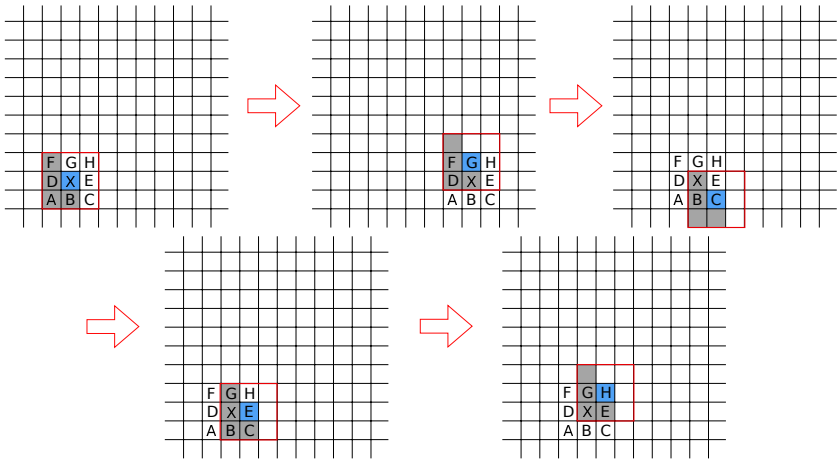


Figure 3.8: For the currently active blue cell only the grey neighbors are checked. We see that if we traverse the grid in the depicted way, we only need to check half of the neighbors for each cell.

We have so far described how to check for potential collisions using a single grid level. When we are dealing with a simulation where there are objects of different sizes we will be using hierarchical grids. The basic procedures that we have explained for a single level still apply in the case of hierarchical grids, what is missing is how to check for potential collisions between objects that are stored in different grid levels. We will start handling this aspect by looking at the related question of how many grid levels are needed in a simulation and how the cell sizes for the grid levels are chosen. As we have mentioned before, we can use heuristics to generate as many grid levels so that the sizes of objects on any level do not differ by more than a certain threshold percentage λ. If we build our grid levels this way we can be sure that there are never too many objects in a cell because of equation 3.2 and that the cell size in each level is well suited to the size of the objects in that level. It is however possible that in some levels of the grid there will only be a few objects, but here the advantages of the hash grid structure pay off because in a hash grid the number of allocated cells is directly proportional to the number of objects in the grid.

Finding Potential Collisions in a Grid Hierarchy

When using a grid hierarchy instead of a single grid the procedure for each level is still the same as illustrated in figure 3.8, but we still need to check in the other levels of the grid for potential collisions. For a grid G with n grid levels with $lmax$ being the level with the largest cell size and $lmin$ being the level with the smallest cell size we could either start the basic procedure at $lmax$ or $lmin$ and then locate the current object in

the lower or respectively higher levels and add potential collision pairs for the objects in the found cell and its neighbors. The exact procedure is described in algorithm 2. An illustration of algorithm 2 is provided in figure 3.9.

Algorithm 2: Procedure for finding potential collisions using a hierarchy of grids

```
Data: hierarchicalGrid G
Result: list<CollisionPair> broadPhaseCollisions
begin
    for i = lmin to lmax do
        foreach cell c in G.level(i) do
            if !c.empty() then
                foreach object o in c do
                    foreach pair (o, x) with x in c or in neighbors(c) do
                        broadPhaseCollisions.pushback((o, x))
                for j = i to lmax do
                    c_o=G.level(j).getCell(o)
                    foreach pair (o, x_j) with x_j in c_o or in neighbors(c_o) do
                        broadPhaseCollisions.pushback((o, x_j))
```

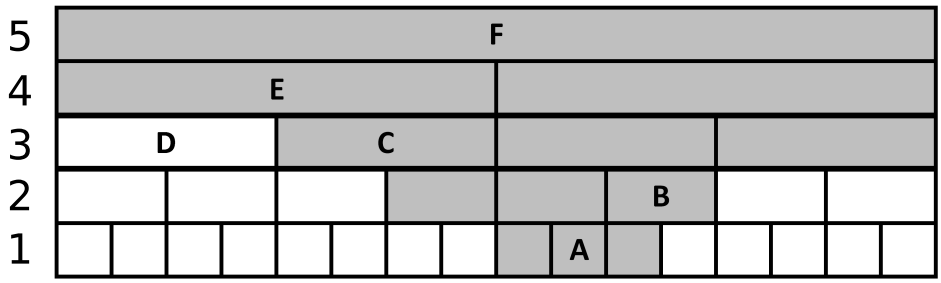


Figure 3.9: 1D Illustration of hash grid search: For object A the grey colored cells are tested and the pairs $(A, B), (A, C), (A, E)$ and (A, F) are added to the list of potential collisions.

Updating the Data Structure

We have so far answered the majority of the central questions about the design and implementation of hierarchical hash grids that we have listed in the beginning of this section. The last missing concept arises from the fact that since we are concerned with simulating moving objects we need to update the positions of our objects in the search grid data structure. In other words, we need to change the object-cell relation in case

the center of an objects bounding sphere has left its old cell and moved into a new cell. The update procedure has to be called in every time step of the simulation. Two approaches for updating the search structure come to mind:

- The search structure is cleared every time that the positions are updated and then all objects are inserted into the search structure again.
- We recalculate for every object the hash value for the current position of the object. We then compare if the newly computed hash value still corresponds to the cell that the objects is currently associated with. If the value is the same then the cell association can remain the same, if the value is different the object has to be deleted from the current cell and reassigned to the cell that corresponds to the new hash value.

The advantages of the simple first approach are that it can be easily implemented and the clearing operation can be executed very efficiently in $\mathcal{O}(m)$ where m is the number of buckets on a given level. The reinsertion procedure is the same as initially filling the search structure and can be done in $\mathcal{O}(n)$ where n is the number of objects in the simulation. For the second approach we only need to compute the hash function and in case that the object gets assigned a different value than in the previous time step, we need to delete it from its current cell which can be done efficiently if the list-based implementation of the hash grid is used and reassign it. Furthermore, we can expect that a large number of objects remains in the same cell since the time step is usually so small that an object only moves a fraction of its size in one time step. Thus, nothing needs to be done for these objects. The second approach then can be handled in only $\mathcal{O}(n)$ which makes it superior to the first approach.

Additional Applications for Uniform Meshes in the Simulation Framework

In this final section of our broad phase analysis we want to discuss the use of our hash grid structure in concrete applications and present alternative choices that may be more effective in some situations. First of all we want to revisit the choice of bounding volume that is used to represent an object in the broad phase. If we are dealing with a standard particulate flow simulation that contains only rigid spherical particles then of the choice of a spherical bounding volume is ideal. For other geometrical shapes or other use cases of search grids a different choice of bounding volume may be more efficient. Another use case of uniform grids in our context is related to the computational mesh of the CFD-simulation. In our framework the mesh used in the CFD-simulation to discretize the simulation domain belongs to class of unstructured meshes. In an unstructured mesh the number of elements that is connected to a vertex of the mesh is irregular. Furthermore, a mapping from spatial coordinates to elements is in general not possible. For calculations that depend on the position of the rigid bodies and the elements of the unstructured mesh that are intersected by the rigid bodies it would often be more efficient if a mapping like in equation 2.1. With such a mapping available

the elements that are intersecting with the rigid bodies can be found efficiently. The unstructured mesh of the CFD-simulation is also hierarchically organized, but usually we are interested in the cells intersecting with the rigid bodies on the finest level of the unstructured mesh. However, even for this level of the unstructured grid the cell sizes can differ when mesh adaptation is used. In this situation a hierarchical uniform grid that contains the elements of the finest level of the unstructured mesh of the CFD-simulation can be used to establish the mapping from a position in space to an element of the mesh. The general procedure of inserting the elements of the CFD mesh into the hierarchical uniform is analogous to that of inserting the bounding spheres of rigid bodies into a grid, but as the elements of the CFD mesh are hexahedral cells this choice of bounding volume may not be ideal. As the hexahedral cells are geometrically more similar to an axis-aligned bounding box (AABB), the choice of an AABB as bounding volume for the elements of the mesh may be advantageous. Also the setup procedure can be slightly modified in case we use AABBs as bounding volumes.
We will now present an alternative setup procedure for hierarchical uniform meshes when AABBs are used as bounding volumes to be inserted into the mesh. A hierarchical uniform grid can be constructed in such a way that the cell size of two successive grid levels differs by a constant factor h. We then can compute the cell size of level i as

$$h_i = h_0 \cdot h^i. \tag{3.3}$$

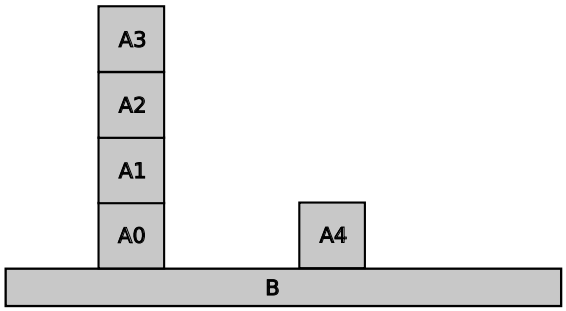
In this setup we insert elements into the uniform grid based on the size of the longest edge of their AABB e_{max}. Elements that fulfill $h_0 \cdot h^{i-1} \leq e_{max} < h_0 \cdot h^i$ are assigned to grid level $i - 1$. A reasonable choice for the constant scaling factor h is $h = 2$. We know that for a finite element mesh even when mesh adaptation is used the difference between the minimum element size and maximum element size is usually bounded by a factor of 10-20. This way a grid hierarchy with at most 5-10 levels is built. The details of how uniform grids can be used to accelerate calculations involving the finite element mesh are given in section 5.4.

3.3 Spatial-Temporal Coherence Analysis

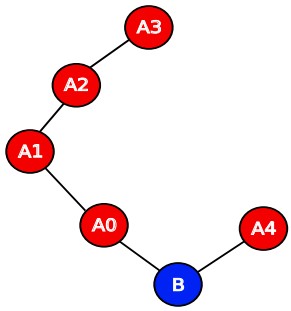
The STC analysis module, as it is termed by Erleben, is acting at different times of the simulation. In earlier simulator designs analysis of spatial-temporal relations between rigid bodies was restricted to fewer functions than those defined in, for example bookkeeping of collision states or caching of contact points. These tasks can be handled between the broad phase and the narrow phase, respectively between the narrow phase and contact determination. This is why this reduced form of spatial-temporal relations between rigid bodies is also called middle phase. The extended form of the middle phase, the STC analysis, acts after the broad phase, after the narrow phase, after contact determination and after collision response to cache contact force results. The STC builds and manages a data structure called contact graph to store the contact information between rigid bodies.

3.3.1 The Contact Graph

The use of contact graphs in rigid body simulation has become more and more wide spread and its functionality was extended to accelerate various calculations. One of the first uses of the contact graph was to find the connected components of the graph. These connected components of the graph correspond to a set of rigid bodies that are in contact, but do not interact with other rigid bodies outside of the connected component. This way in the contact force computation these connected components can be handled as independent units and the problem can be broken down into smaller independent problems. Another use of contact graphs is found in shock-propagation [80, 26]. We refer to shock-propagation as the forwarding of a contact force that arises when a rigid body collides with a connected set of rigid bodies. The contact graph is used to arrange the pairwise numerical contact force calculation in such a way that it propagates the force through the set of rigid bodies in a way that corresponds to the geometric arrangement of the rigid bodies (see figure 3.11). In our implementation we use a contact graph to compute connected components, relational collision states, store geometric contact points and cache contact forces which can be used as an initial solution for the contact force solver to speed up computation.
Formally, a contact graph is an undirected graph $G(V, E)$ where the set of nodes V corresponds to the rigid bodies in the simulation. Relational information between rigid bodies are represented by the set of edges E of the graph. An edge between two nodes of the graph is added when the broad phase collision detection reports a potential collision or a close proximity, meaning the two rigid bodies are located in the same of neighboring cells of the broad phase search grid (see figure 3.10).



(a) Dynamic bodies stacked onto a fixed body



(b) Contact graph corresponding to the rigid body configuration

Figure 3.10: A contact graph example: We see a fixed rigid body with some dynamic rigid bodies stacked onto it. In the corresponding contact graph we use red nodes for dynamic bodies and blue nodes for fixed bodies.

Furthermore, we see that the nodes of the contact graph can have different properties. In figure 3.10 we see so called *dynamic rigid bodies* which are rigid bodies that are allowed to move freely and *fixed rigid bodies* which are in our context as boundaries

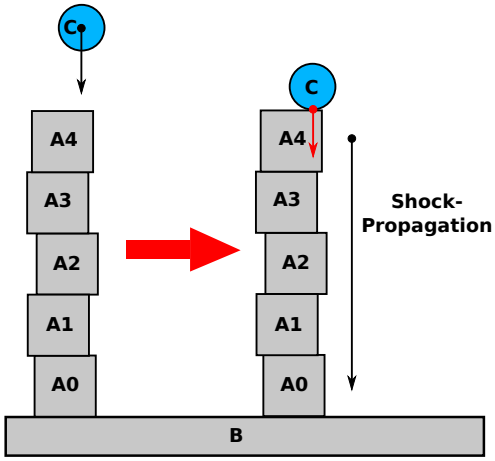


Figure 3.11: A stacked heap of rigid bodies is resting on a fixed rigid body B. When the heap is hit by body C a force is applied and propagates through the heap. Most collision force solvers show better convergence behavior when the collision forces per pair are calculated in an order that corresponds to their geometrical configuration (here: (C,A4), (A4,A3), (A3,A2), (A2,A1), (A1,A0), (A0,B)).

of the computational domain or as solid immovable obstacles in the domain. Just as nodes also the edges of the contact graph can have different properties, an edge can be a broad phase close proximity, a newly detected colliding state or a touching state that already persists over the course of many time steps.

Building the Contact Graph: Insertion and Removal of Edges

As we have said before the STC module with its main data structure of the contact graph is activated at several stages of the simulation. The first activation is after the broad phase when the list of current potential collision pairs is available. The contact graph is updated with this broad phase information which is useful because edges can be directly removed if the broad phase collision detection revealed that in the current time step the involved rigid bodies are at a sufficiently large distance from each other to not be part of a collision. The broad phase considers two bodies to be sufficiently distant when they are more than one cell of the broad phase grid. Although this criterion is just a simple heuristic, it proves to be reliable in practice because our simulations are usually set up in a way that objects only move a fraction of the cell size per time step. The removal of edges at this early stage is useful to reduce the iteration range in the following calculations that involve iterations over all edges. Potential collision pairs found by the broad phase that are not present in the contact graph are added as new edges into the graph. Furthermore, we might not directly want to remove edges in this stage in case the bodies where in contact in the not too distant past. The reasoning behind this is that the bodies can come into contact again and for this case we still want to have the cached contact information available.

Connected Components and independent Contact Groups

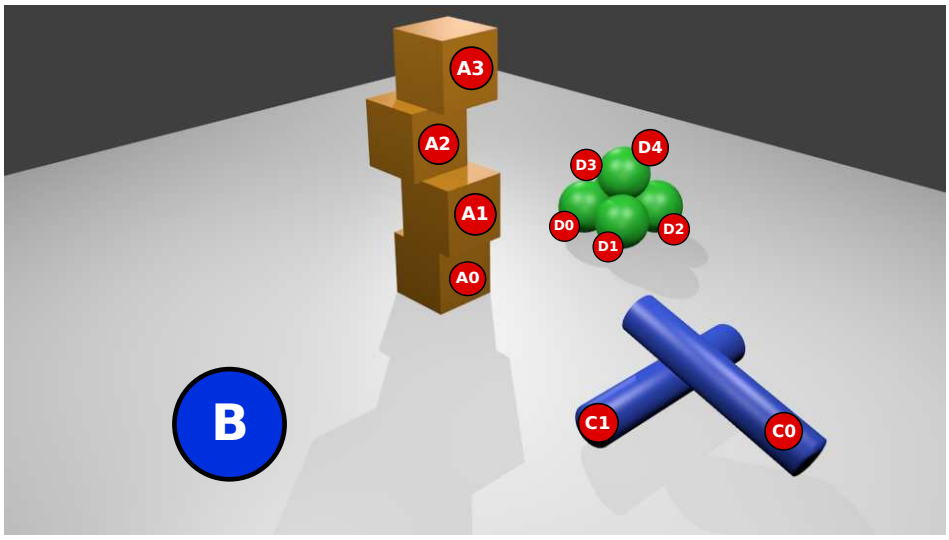
A *contact group* is a set of rigid bodies that is linked by contacts between the rigid bodies. In the contact graph such a configuration corresponds to a connected component in the graph. The interesting properties of contact groups is that they represent a closed, independent unit when computing contact forces, meaning the rigid bodies in one contact group do not interact with rigid bodies from other contact groups. We thus can break down the computation of contact forces into smaller independent units. Since a contact group is basically a connected component in the contact graph they can be calculated by classical procedures to find connected components with some minor modifications. The need to modify the connected component algorithm arises from the fact that if a link between two or more otherwise not connected components is established by an edge between a dynamic rigid body and a fixed rigid body, this edge can not be used in the connected component algorithm and we treat the respective components as not connected. We see an example of independent contacts groups in figure 3.12 and the modified connected component procedure in algorithm 3. The slight modifications notwithstanding the complexity of the modified connected components algorithm remains $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$.

Algorithm 3: Entry Point function of the connected component search.

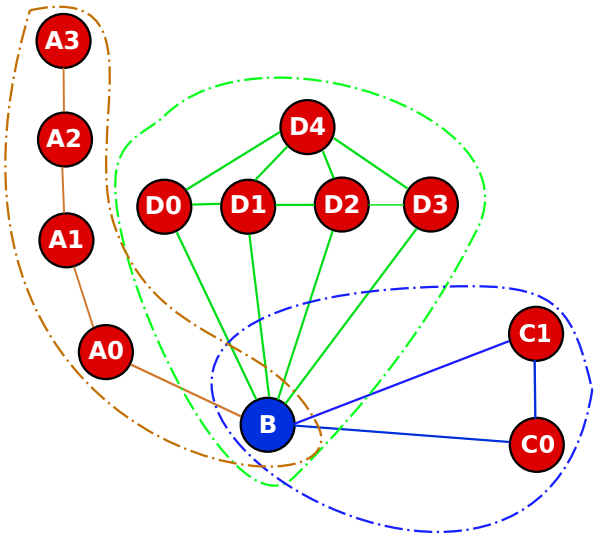
Data: Contact Graph $G(V, E)$
Result: For all $v \in V$ the number of the connected component is assigned
begin
  $group = 0$
  foreach *node* v *in* V **do**
    $v.contactGroup = group$
  foreach *node* v *in* V **do**
    if $v.contactGroup == 0$ **then**
      $group = group + 1$
      depthFirstSearchCG(v,$group$)

Procedure depthFirstSearchCG(v,group)

begin
  $v.contactGroup = group$
  foreach w *in* $v.getNeighbors()$ **do**
    if $w.rigidBodyType != fixed$ **then**
      if $w.contactGroup == 0$ **then**
        depthFirstSearchCG(w,group)



(a) Example configuration of different contact groups



(b) Contact graph with the different contact groups of the example configuration

Figure 3.12: We see groups of rigid bodies on top of the fixed rigid body B. We see three different contact groups: the green objects, the blue objects and the dark orange objects, each group including the fixed body B. Although there is a path in the contact graph between the groups, it is ignored because we encounter the fixed body B on the path.

Benefits and Usage of Contact Groups and Graphs

Contact groups can be used when computing the contact forces, depending on the type of the contact solver these can have a runtime complexity that is non-linear. Especially

in these cases breaking down the contact force problem into smaller independent units can be beneficial to avoid larger problem sizes. The independent units of the contact force problem are exactly the different contact groups.
The contact forces calculated in one step of the contact solver are stored for each collision pair in the edges of the conact graph. When in the next iteration of the simulation loop the same collision pair is reported again, the contact forces computed in the previous iteration can be used to warm start and to accelerate the convergence of the contact solver [25].
But not only contact forces are cached in the edges of the contact graph, also narrow phase contact point information is available in the edges of the contact graph. Another optimization possibility is to use this information to reduce the expensive geometric calculations that are done in the narrow phase or at least to make these geometric calculations easier. Since we have available the contact points, contact normals, the positions in the current and previous time step, we can test for example if the contact normal is still well enough oriented in the current time step to valid and then skip this computation in the current step and reuse the old data. When thinking in terms of the STC module and its main data structure the contact graph, these steps are the part of the simulation where the STC module is called during the narrow phase and before the contact determination. We see that the STC module is updated and called for optimization purposes at different stages of the simulation as we have outlined in the beginning of this section. A schematic overview of the simulation loop of the rigid body simulator with all its modules we discussed so far is depicted in figure 3.13.

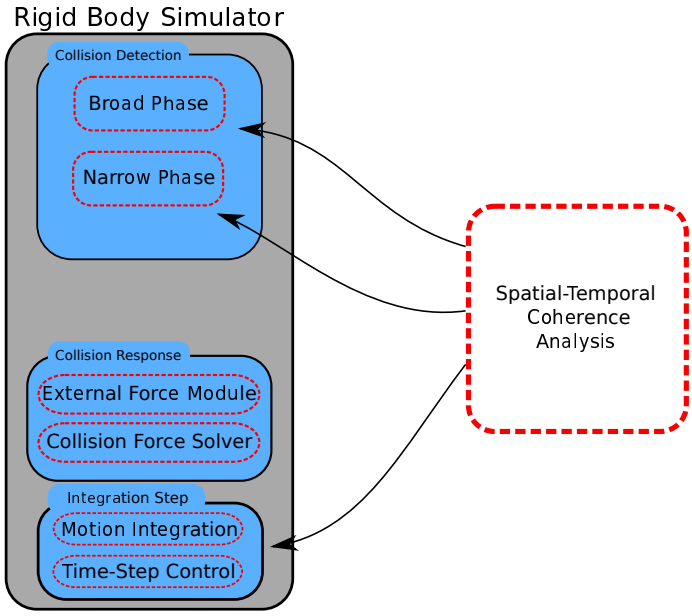


Figure 3.13: Activation of the STC at various stages in the simualtor.

3.4 The Narrow Phase Module

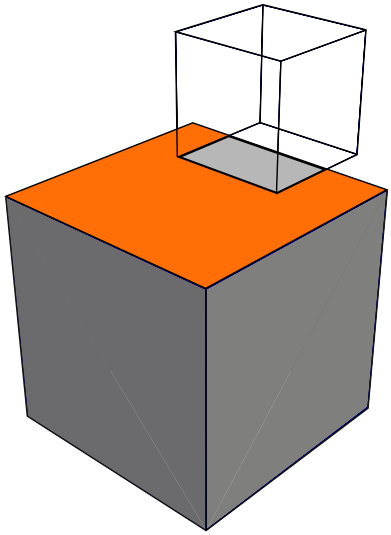
The narrow phase module's position in the simulation loop of the rigid body solver is directly before the contact solver. The reason for this is that basically all collision force models require geometric information about the collision between the rigid bodies in order to compute the resulting collision forces. The challenging aspect of the narrow phase is to provide efficient, accurate and reliable methods to compute the geometrical information needed for the contact solver for arbitrary geometries. The reliability aspect aims at the property of a particular narrow phase algorithm to guarantee a result, as we do not want to rely on fallback procedures and unnecessary computations.
In our simulation we set our aim to be able to simulate arbitrarily shaped rigid bodies and their interaction. For this we need a way of representing simple particles, complex particle shapes, various obstacles in the domain, simple and complex domain boundaries as well as complex rigid body geometries. With the goal being to be able to represent arbitrary geometries in our simulation we consider using the following geometry representations for different configurations:

- Spheres: simple particles, obstacles, etc.
- Convex analytical shapes: cylinders, boxes, ellipses, conical sections, etc. for particles or obstacles
- Planes, boxes: simple domain boundaries, obstacles
- Surface triangulations: complex shapes, complex domain boundaries.

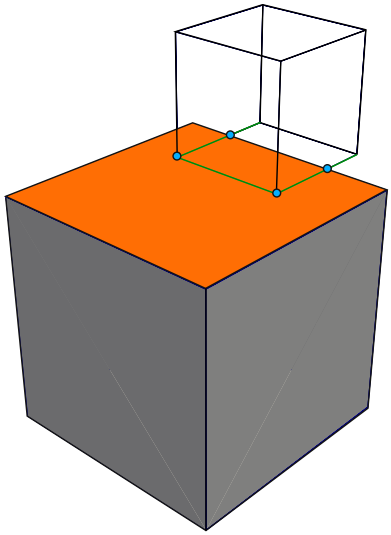
Another important question concerning the narrow phase is how a collision between rigid bodies is actually described geometrically. In the simple case of two spherical particles colliding this question is trivial. Let us assume we are dealing with two touching spherical particles s_0, s_1 with center positions $\mathbf{x}_0$, $\mathbf{x}_1$ and radii r_0, r_1. Then the normal of the contact is $\mathbf{n} = |\mathbf{x}_1 - \mathbf{x}_0|$, the contact points are $\mathbf{p}_0 = \mathbf{x}_0 + r_0\mathbf{n}$ and $\mathbf{p}_1 = \mathbf{x}_1 - r_1\mathbf{n}$. So in this simple case the geometric contact information consists of the contact normal, the distance (which is zero in the case of touching spheres) and a contact point on the surface of each spherical particle. For more complex scenarios the situation is more difficult as figure 3.14 illustrates.

3.4.1 Contact Sets

As we have shown, the intersection between two complex 3-dimensional geometries is often a complex entity like a surface or in the case that there is an overlap even a volume. The real full analytic description of an intersection we call the *full contact set*. However, computing collision forces with a full contact set is a complex and expensive computation. The contact laws that we will be using operate on a *reduced contact set* that consist of discrete contact points, while still these contact laws offer very good accuracy and can be computed rapidly and efficiently. Reducing the contact



(a) Full surface for a box-box intersection



(b) Reducing the contact information to discrete points

Figure 3.14: The full intersection between the two boxes in (a) is a surface, for efficiency purposes this full contact set is reduced to the intersection of the top quadrilateral of the box with the end points of the edges of the other box.

set is often done by using only the points at the boundary edges of the contact surface instead of using the contact surface (see figure 3.14). While for most contact laws we only need to compute a set of contact points with corresponding contact normals it is often beneficial to compute and store additional information that will allow us to accelerate computations in other stages of the simulation, especially considering that we will get the additional information along the way of computing the principal contact information. So to summarize we compute the following contact information:

- an ID identifying the two rigid bodies on the collision pair,
- the contact points on the surface of the rigid bodies,
- a contact normal for each contact point,
- the distance between the rigid bodies (which does not have to be zero),
- optionally for slightly penetrating contacts: the penetration depth.

In the following we will focus on the question of how to compute the contact information that we have just outlined and the choice of algorithms for the different geometry representations. We have mentioned before that cached contact sets can be used to accelerate the computation of contact information, we will at first discuss the general case when no stored contact information is available. The input to our narrow phase

algorithms comes from the broad where we added potential collisions pair in case they are in the same cell or in directly adjacent cells of the broad phase grid and their bounding volumes do overlap. The fact that the bounding volumes of two rigid bodies overlap does not yet prove that there is a collision (see figure 3.15). To finally determine whether a collision occurred or not, we need a different criterion which could be a non-empty intersection or a very small or even negative signed distance.

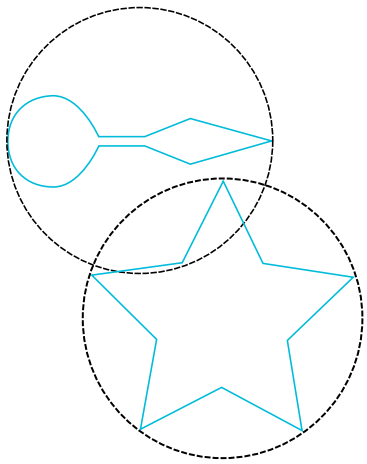
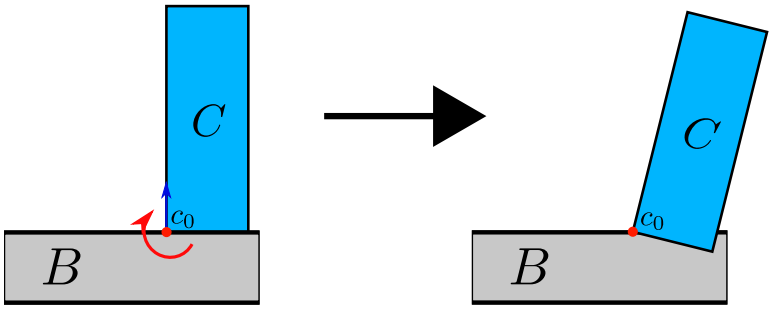


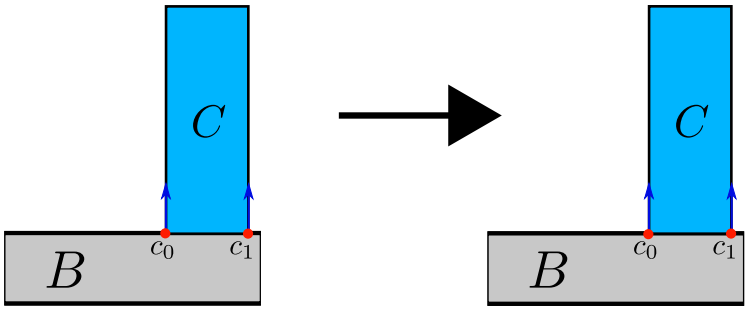
Figure 3.15: The bounding volumes of our objects overlap, but the real geometries do not collide.

3.4.2 Distance Computation

The first step in determining whether we consider two rigid bodies from a broad phase collision pair as colliding is checking the minimal distance between them. Computing minimal distance between convex shapes can be done very efficiently using the *Gilbert-Johnson-Keerthi* algorithm (GJK) [29, 24]. The GJK-algorithm does a good job of finding the pair of points on the surfaces of two convex shapes for that the distance is minimal. Passing just the single pair of closest points as a contact set to the contact solver is incorrect in many cases. In the examples in figure 3.14 or in the stacked boxes in figure 3.10 we need at least the end points of the edges, the reduced contact set, in order to get a valid solution from the contact solver. If the contact force would be applied in only a single contact point in these cases it would cause the box to rotate (see figure 3.16) and the box stack to collapse while in reality a stable box stack would be the physical solution. The advantage of the GJK-algorithm is however that it yields a definite result to our question whether we should consider the pair of rigid bodies to be colliding or not. Either the minimum distance tells us that the rigid bodies are separated by a distance that is large enough so that we do not need to consider the pair in our contact solver or it tells us that there is an intersection. We then need a way to enrich the solution produced by the GJK-algorithm to yield a contact set that will produce a valid result in the contact solver. For boxes this could be the Voronoi Clip



(a) Single contact point point configuration



(b) Stabilization by multiple contact points

Figure 3.16: A box C is resting on top of box B. In order for the contact solver to produce correct results, we need to compute an appropriate contact set. If only a single contact point is generated, we introduce a rotation of C into the box B because the information passed to the contact solver is too limited to fully describe the contact configuration. When two contact points are given the contact solver is able to balance out the contact and gravity forces to produce a stable configuration.

(V-Clip) algorithm by Mirtich [54] or variants of it. In our simulator complex non-convex shapes are represented by surface triangulations which can potentially consist of tens or hundreds of thousand triangles. These numbers basically require algorithms for distance or contact set computation that have efficient runtime complexity with regard to the number of triangles. Our solution for these cases is to adopt the concept of signed distance map which appears in literature in the works of Guendelman [37]. Signed distance maps are a classical trade-off that offers favorable algorithmic complexity at the cost of increased memory usage. A signed distance map is basically a cartesian grid around the surface triangulation. In a preprocessing step the distance for all the points in the distance map can be calculated using any distance algorithm for meshes. Then in the simulation we can transform the shapes into the coordinate system of the signed distance map, locate the points in $O(1)$ and get the distance via trilinear interpolation. We can additionally store normals in every point of the distance map and also produce valid contact normals using interpolation. If we set up the distance maps like we have explained they provide all the necessary information to compute

distances and contact sets and contact normals for collisions between mesh geometries and any other geometry representation. The downside of using distance maps are additional memory costs and that the accuracy of the distance computaion is dependent on the resolution of the distance map, so higher accuracy comes at the price of increased memory. At this stage we have summarized all the tools we need to forumlate the general narrow phase procedure: As we see in algorithm 4 we first check whether we

Algorithm 4: General Narrow Phase procedure

```
Data: list<CollisionPair> broadPhaseCollisions
Result: list<CollisionPair> narrowPhaseCollisions
begin
    foreach CollisionPair p in broadPhaseCollisions do
        if p.boundingVolumes.overlap() then
            if evaluateContactCondition(p) then
                narrowPhaseCollisions.pushback(p)
```

are dealing with a broad phase collision pair (a pair whose bounding volumes overlap) or a pair of rigid bodies that satisfied the *contact condition*. The contact condition tells us whether we should consider the pair in the contact solver. The simplest cases of the contact condition are when the rigid bodies are touching or slightly intersecting, for these cases we directly compute the contact information. More complicated are those cases where the rigid bodies are not exactly touching or penetrating, but are very close. It is advisable to add those pairs to the contact solver or otherwise we would run the risk of penetrations when stepping from one time step to the next when the contact solver determines forces that cause two rigid bodies to move closer together without having them as a collision pair. The details on how to formulate the contact condition are given in chapter 4. An example configuration of when a pair of rigid bodies should be added as a contact pair in the collision solver is when they are very close is shown in figure 3.17. The methods we have mentioned so far enable us to compute distances and contact information for the different geometric shapes and geometry representations we have mentioned in the beginning of this section, we can summarize these as follows:

- Simple particles: analytic solution,
- convex/convex or convex/particle: GJK + Minkowski portal,
- Particles/meshes/boundaries: analytic or distance maps,
- mesh/convex: distance map
- mesh/mesh: distance map

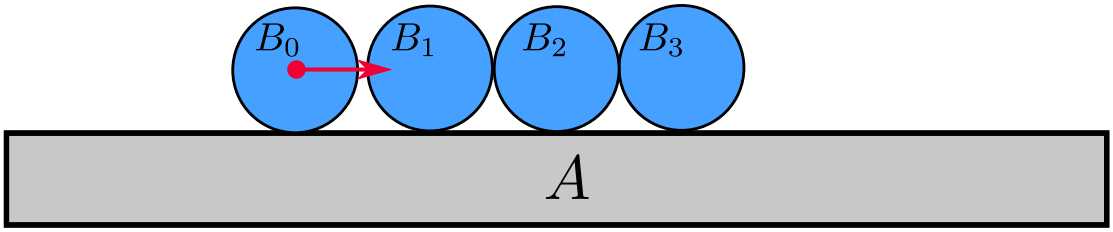


Figure 3.17: We see a situation where three spheres B_1, B_2, B_3 are in touching contact. The sphere B_0 is moving on collision course to the other spheres, beginning with sphere B_1, but a small separation distance remains. If the velocity and the time step are configured in such a way that the sphere B_0 travels more than the separation distance in the current time step, we would get an interpenetration. If this is the case we can add a contact pair (B_0, B_1) in order to prevent a penetration and trigger the computation of contact forces in the current time step.

Distance Maps

As we have seen distance maps play an important role especially when handling more complex geometries, so we will take a look at their definition, generation and application in more detail. A distance map consists of a rigid body geometry that is represented as a surface triangulation and an axis-aligned boundary box for the surface mesh. The axis-aligned bounding box is then expanded by a certain threshold distance for that we want to compute distance information. This step would be redundant if we only wanted to compute distances for collision checking as for any point that is farther away from the geometry as the bounding box we would not need distance information, but as we need distances for other computations in our simulation framework the expansion of the bounding box is needed. We then create a structured cartesian grid in the bounds defined by the bounding box of surface mesh. We proceed to compute distance information for every point in this grid and store this information in the vertices of the grid. If we then want to compute the distance of a query point $\mathbf{x}$ to the geometry we would locate the point in the structured grid (the distance map) and read the distance stored in the nearest vertex of the grid or compute it by trilinear interpolation [43]. In order to use these distance maps for moving rigid bodes, the distance map is calculated in the local space of the rigid body, the space whose origin is the center of mass of the rigid body and whose axes correspond to the principal axes of the rigid body. Since we store the transformation matrices $\mathbf{R}_i$ of our rigid bodies at all times, we can transform points $\mathbf{x}_j$ that define our shapes into the coordinate system of the rigid body and then perform our distance map point location in the local coordinate system of the rigid body:

$$\mathbf{R}(\mathbf{x}) = \mathbf{R}_i^T \cdot \mathbf{x}.$$

In order to compute the distance map in the preprocessing step we can use methods based on AABB trees or GPU-based techniques, if time is of no concern in the prepro-

cessing then even brute force approaches can be used. The final procedure to compute distance maps is outlined in algorithm 5.

Algorithm 5: Procedure for building a distance map for a geometry

```
Data: Geometry g, threshold λ
Result: DistanceMap map
begin
    box = g.getBoundingBox()
    box.size += λ
    map = createStructuredGrid(g,box)
    foreach vertex v in map do
        p = v.getCoordinates()
        v.distance = computeDistance(g,p)
```

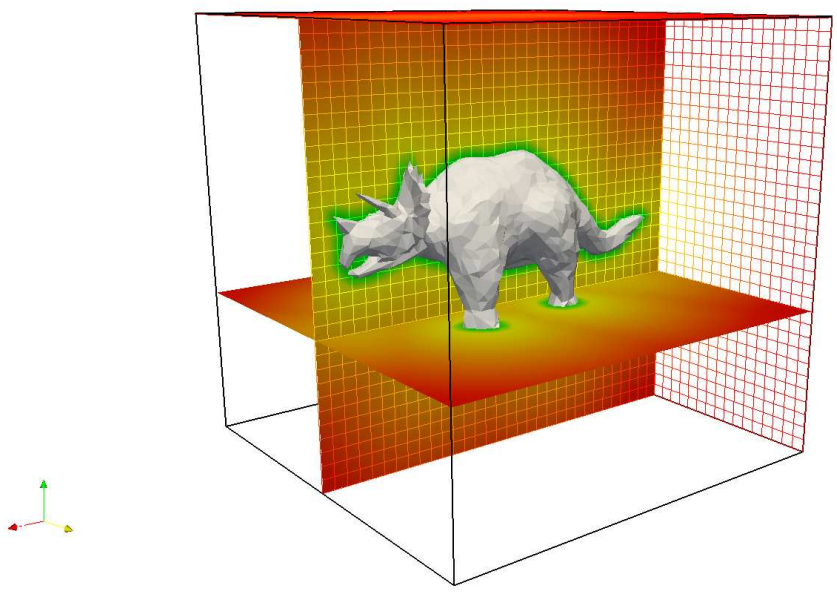


Figure 3.18: Example of a distance map for a surface mesh: The distance map is a structured grid that stores precomputed information like distances and normal vectors to the surface. The precomputed information will then be available for calculations during the simulation.

Distance maps for complex objects do not only provide an efficient way to calculate collision information between complex geometries, but they can also be used for the computation of distance fields on the grid. Distance fields in our context are mainly used to control mesh deformation techniques in order to concentrate mesh vertices near the surface of an object. The mesh deformation algorithms need precise distance information so that the vertices can be relocated at the correct positions, since the distance

map provides precise distances in a predefined vicinity around the object, they are well suited for accelerating local deformation techniques that operate on the vertices in a certain distance around the object and pull these vertices closer to the surface. In these local deformation schemes precise distances for vertices that are outside of the local area of interest are not needed, so for these vertices an approximation of the distance is sufficient for example the distance to the axis-aligned bounding box of the object.

Another data structure that deserves attention in the context of calculating collisions between complex objects are inner sphere (see figure 3.19) representations of complex geometries [96, 84, 94, 97, 95]. These data structures represent the volume of comples objects by a set of inner spheres. In some of the construction procedures the inner spheres are non-overlapping which reduces the amount of spheres used to represent the object. The advantages of inner sphere representations are that the point classification used in the interface tracking of the CFD-simulation can be handled in a complexity that is independent on the number of vertices in the grid if a mapping from spatial coordinates to grid elements is available, because we need to test only the vertices that are located inside the spheres that make up our object, all that we need is a procedure to access the vertices of the grid that are inside the spheres in constant time. How to build the data structures that provide this feature we have explained in previous sections. The disadvantage of inner sphere representations to distance maps is that they do not provide us with precise distance information around the surface of the object because of the sphere representation. Furthermore, inner sphere representations need to be transformed as the objects change position and orientation, and because of this cannot make use of precomputed information that can be read in constant time as is the case with distance maps. So in a simulation where distance fields are needed the distance maps have a clear advantage over inner sphere representations, if fast point classification is more important then an inner sphere representation may be more suitable. It is possible to easily create inner sphere representations on the basis of a distance map by choosing the bounding spheres of the cells of the distance map that are located inside our geometry.

Data structures such as distance maps and inner sphere representations map very well to GPU implementations, the structured grid can either be implemented as a 3D texture or straightforward as a CUDA array [16, 60]. For the distance map all that is required is a transformation of a point by matrix multiplication followed by a value lookup. These kinds of operations are extremely efficient on the GPU and can be executed in parallel by different GPU threads. An inner sphere representation would be a CUDA array of sphere structures and additionally the transformation matrix and the vertices of the computational mesh are needed. A GPU implementation for inner sphere would then assign threads to each sphere and each thread would perform the test which points of the mesh are inside the radius of the individual sphere.

Figure 3.19: Inner Sphere Representation: the volume of the object is approximated with spheres of different size. Several inner sphere construction methods exist, the displayed version was produced by the Coll-Det software package [15, 95].

Narrow Phase Acceleration: Contact Caching

In the preceeding section we several times mentioned the caching (storing between time steps) of contact information in order to accelerate narrow phase computations. Distance computation via the GJK algorithm can easily be accelerated by storing the pair of vertices with minimal distance. A quick recap of the GJK algorithms brings to mind that it iteratively generates points on the surface of the pair of rigid bodies that converge to the minimal pair of vertices. If we provide an initial solution to this procedure using the pair of vertices we have cached from the previous time steps the number of iterations that it takes the GJK to converge can usually be drastically reduces, in many situations we can even expect it to converge in one step.

3.5 The Contact Generation Module

The contact generation module is closely connected with the broad phase module. The narrow phase algorithms determine whether we consider a broad phase collision pair as a colliding contact in the contact solver and hence it computes a minimum distance between the rigid bodies and the vertices with the minimum distance as contact points. With the help of these vertices a contact normal can be calculated. Depending on the algorithm and the involved geometries the narrow phase sometimes already does the work of the contact generation module for example in the case of simple spherical particles colliding all contact information is already available after the narrow phase. For box-box collision Mirtich's V-Clip algorithm [54] or its variants are used then also all contact information is available. For some geometries however some work still remains to generate a (reduced) contact set that produces viable results in the contact solver. Especially when the narrow phase uses the GJK for distance calculation the

output of a pair of vertices with minimum distance is often not enough for the contact solver to produce a valid result. For these cases the contact determination module is called to generate additional contact points and contact normals to provide the contact set that is needed by the contact solver which is usually the reduced contact set that consists of the end points of the edges of the intersection surface. In figure 3.20 we illustrate a configuration where the single contact point on each surface is not enough to produce a contact set that will satisfy the contact solver. This is why our contact module contains methods to compute intersections between the various shapes used in the simulation and processes them in such a way that we can obtain a proper reduced contact set. Generally a contact determination step is needed when our narrow phase algorithm does not already include a proper reduced contact set which is our case is mainly the case for GJK outputs from general convex shape collisions. An overview of how to find efficiently calculate intersections for these shapes can be found in the works of Eberly [76, 22] or Erikson [24]. Still we have to answer the question what

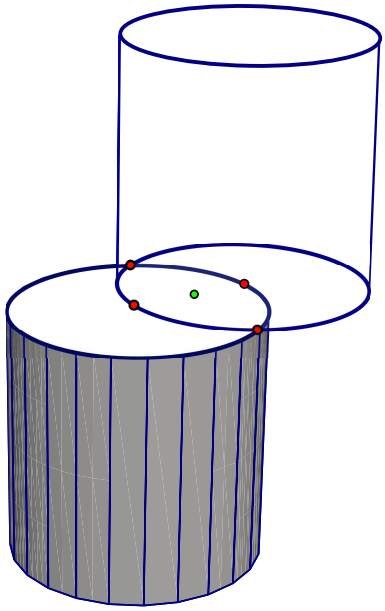


Figure 3.20: The GJK algorithm only generates one contact point in its standard form that is based on the minimum distance (the green point for example). Applying the contact only in that point would not result in a stable configuration. Generating contact points on the edges of the intersection surface helps to stabilize the configuration and produce the desired result where one cylinder is resting on top of the other cylinder.

happens when the narrow phase considers a pair of rigid bodies colliding, but in fact they are still separated by a small distance ϵ. In this case we have the 'collision' normal available from the two points with the minimum distance on the surface of each body. We can then artificially produce an intersection by projecting the surface of one rigid body onto the surface of the other one in the direction of the contact normal. This enables us to use our standard algorithms for intersection and computation of contact

information.

Chapter 4

Efficient Contact Solvers for Rigid Body Simulation

In this chapter will focus on the module that is responsible for producing a physical response to collisions between rigid bodies in our simulation. We will present different models to implement a contact solver, analyze them and discuss their strengths, shortcomings and their preferred area of application. In order to build a solid foundation to introduce the theory of collision models, we summarize the basic concepts from physics that are necessary for an understanding of collision models. For a more detailed and extended treatment of these concepts from physics and classical mechanics we would like to direct the reader to the works of Goldstein [34] or Kuypers [46].

4.1 Particles

A particle is an object with a constant mass m, a position $\mathbf{x}$ and an orientation θ. These properties make the particle the simplest object (an *idealization*) that allows the study of motion and are the reason why a particle is often referred to as a *point mass*. If a particle moves, some of its properties change and are thus dependent on time.

4.1.1 Equations of Motion for Particles

The time dependent quantities that determine the motion of a particle over time are the position $\mathbf{x}(t)$, the velocity $\mathbf{v}(t)$ and the acceleration $\mathbf{a}(t)$. These are related in the following way:

$$\mathbf{v}(t) = \frac{\partial \mathbf{x}(t)}{\partial t} \tag{4.1}$$

$$\mathbf{a}(t) = \frac{\partial \mathbf{v}(t)}{\partial t} = \frac{\partial^2 \mathbf{x}(t)}{\partial t} \tag{4.2}$$

According to Newton's second law of motion particle acceleration is caused by the force $\mathbf{F}(t)$ acting on the particle. When studying the motion of a particle it is then

necessary to calculate the *total force* or *net force* $\mathbf{F}_{net}(t)$ acting on a particle which is the sum of all individual forces $\mathbf{F}_i(t)$ acting on the particle. The relation between Newton's second law of motion, the net force $\mathbf{F}_{net}(t)$ and the particle acceleration $\mathbf{a}(t)$ is as follows:

$$\mathbf{F}(t) = m\mathbf{a}(t) \tag{4.3}$$

$$\mathbf{F}_{net}(t) = \sum_i \mathbf{F}_i(t) \tag{4.4}$$

$$\mathbf{a}(t) = \frac{\mathbf{F}_{net}(t)}{m} \tag{4.5}$$

4.1.2 Particle Rotation

The angular motion of a particle in 3D is described by a vector $\boldsymbol{\omega}(t)$ that points into the direction of the axis the particle rotates around and the speed of rotation. Analogously, $\boldsymbol{\theta}(t)$ defines an axis of rotation and an angle around it to describe the current orientation of the particle. The SI physical unit for angular velocity is *radians per second*, for different purposes it is useful to be able to convert this into the linear velocity which can be done using the following relation:

$$\mathbf{v}(t) = \boldsymbol{\omega}(t) \times \mathbf{r}(t), \tag{4.6}$$

here $\mathbf{r}(t)$ is a vector from the center of rotation to the particle position $\mathbf{x}(t)$. The resulting linear velocity vector is normal to both the axis of rotation and $\mathbf{r}(t)$ following from the cross product properties. The geometric interpretation of this relationship is depicted in figure 4.1. The magnitude of this linear velocity $\mathbf{v}(t)$ is

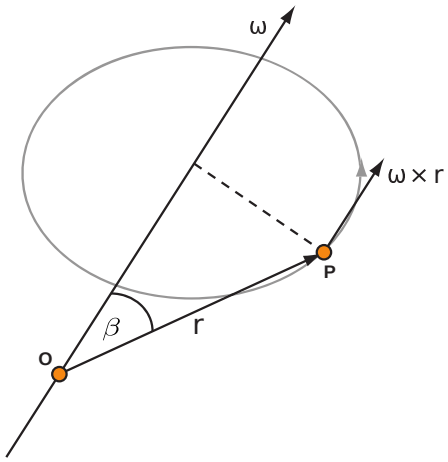


Figure 4.1: Rotation of a particle around an axis

$$|\mathbf{v}(t)| = |\boldsymbol{\omega}(t)|\,|\mathbf{r}(t)|\,sin(\beta), \tag{4.7}$$

where β is the angle between $\boldsymbol{\omega}(t)$ and $\mathbf{r}(t)$ with $|\mathbf{r}(t)|\, sin(\beta)$ being the part of the displacement vector $\mathbf{r}(t)$ that is normal to rotation axis. The relation of angular acceleration to angular velocity is similar to that of linear acceleration to linear velocity:

$$\alpha(t) = \frac{\partial \boldsymbol{\omega}(t)}{\partial t} \tag{4.8}$$

$$\mathbf{a}(t) = \alpha(t) \times \mathbf{r}(t) \tag{4.9}$$

The force that actually causes a particle to rotate around an axis of rotation is the *torque* $\boldsymbol{\tau}(t)$:

$$\boldsymbol{\tau}(t) = \mathbf{r}(t) \times \mathbf{F}(t), \tag{4.10}$$

so the vector $\boldsymbol{\tau}(t)$ is normal to $\mathbf{r}(t)$ and $\mathbf{F}(t)$ and thus points in the direction of the axis of rotation.

4.1.3 Linear and Angular Momentum

The product of the mass and velocity of a particle is called *linear momentum* $\mathbf{p}(t)$:

$$\mathbf{p}(t) = m\mathbf{v}(t), \tag{4.11}$$

from this relation follows that a force which causes a change of particle velocity also induces a change of linear momentum:

$$\mathbf{F}(t) = m\mathbf{a}(t) = \frac{m\partial \mathbf{v}(t)}{\partial t} = \frac{\partial \mathbf{p}(t)}{\partial t}. \tag{4.12}$$

A change in linear momentum is referred to as *linear impulse* and is defined as constant force $\mathbf{F}$ acting over a certain period of time Δt:

$$\mathbf{J} = \int \mathbf{F} dt = \int m\mathbf{a} dt = \int m\frac{d\mathbf{v}}{dt} dt = \int d\mathbf{p}. \tag{4.13}$$

In many models for the forces arising from a collision between particles or rigid bodies the impulse is the physical quantity that is computed in order to induce the velocity change arising from the collision. So the constant force that is acting over a period of time Δt (the collision time) can be regarded as the collision force that resolves the collision. The angular equivalent to linear momentum is the angular momentum $\mathbf{L}(t)$ which is related to the linear momentum in a similar way as force is related to torque:

$$\boldsymbol{\tau} = \frac{\partial \mathbf{L}(t)}{\partial t} \tag{4.14}$$

$$\mathbf{L}(t) = \mathbf{r}(t) \times \mathbf{p}(t). \tag{4.15}$$

Just as the linear impulse is used to calculate collision forces, an *angular impulse* that represents the rotational part of the collision force is used in its determination. The angular impulse is a change in angular momentum and is determined by a torque force acting over the collision time Δt:

$$\boldsymbol{\tau}\Delta t = I\Delta\boldsymbol{\omega} = \Delta\mathbf{L}. \tag{4.16}$$

Here I is the particle inertia which will generalize to a tensor for general rigid bodies.

4.1.4 Work, Energy and Kinetic Energy

When a force is acting on an object and causes a displacement of this object then *work* is done. If the object moves a distance d in direction $\mathbf{s}$ an amount of work ΔW is done, this amount of work can also be expressed as the product of force and velocity over the time it takes to travel the distance d:

$$\Delta W = \mathbf{F} \cdot d\mathbf{s} = \mathbf{F} \cdot \mathbf{v}\, dt, \tag{4.17}$$

over the whole trajectory of the object this leads to:

$$W = \int_{t0}^{t1} \mathbf{F} \cdot \mathbf{v}\, dt = \int_{\mathbf{x}(t0)}^{\mathbf{x}(t1)} \mathbf{F} \cdot d\mathbf{s}. \tag{4.18}$$

The *energy* of a particle is its stored capability to do work, the *kinetic energy*:

$$E = \frac{1}{2} m \left|\mathbf{v}\right|^2, \tag{4.19}$$

which is important for modeling collisions as elastic of plastic collisions. In an elastic collision the kinetic energy is conserved and in an elastic collision a part of the kinetic energy is consumed.

4.2 Rigid Bodies

Having introduced the basic physical concepts for particles, we can start focusing on rigid bodies. In quantum mechanics a rigid body is defined as a system of particles or point masses in which the distance between each possible pair of different particles is invariant in time. According to this definition of rigid bodies the mass of a rigid body can be calculated by summing up the individual point masses:

$$m = \sum_i m_i \tag{4.20}$$

A concept of great importance for the study of rigid bodies is the *center of mass*:

$$\mathbf{x}_{cm}(t) = \frac{\sum_i m_i \mathbf{x}_i(t)}{\sum_i m_i} = \frac{\sum_i m_i \mathbf{x}_i(t)}{m} \tag{4.21}$$

The importance of the center of mass can be illustrated in the case of external forces acting on the rigid body. When an external force acts on a rigid body it causes the body to move and the position of $\mathbf{x}_{cm}$ changes. The external force can be calculated as the sum of the forces on the particles that make up the rigid body. This total force can then be used to calculate the acceleration of the center of mass which can be used to obtain

the new position of the center of mass:

$$\mathbf{F}(t) = \sum_i m_i \mathbf{a}_i(t) = \sum_i m_i \frac{d^2\mathbf{x}_i(t)}{dt^2} \tag{4.22}$$

$$= \frac{d^2}{dt^2} m\mathbf{x}_{cm}(t) = m\mathbf{a}_{cm}(t), \tag{4.23}$$

where $\mathbf{a}_{cm}$ is the acceleration. The same is true for other quantities like the linear momentum and the angular momentum. While the definition of a rigid body as a collection of discrete point masses is useful, there is a definition in classical mechanics that describes a rigid body as a continuous mass distribution over its volume:

$$m = \int_V \rho(\mathbf{x})\, dV, \tag{4.24}$$

where $\rho(\mathbf{x})$ is the density function of the body at position $\mathbf{x}$. In the continuous definition of a rigid body the center of mass is given by the volume integral:

$$\mathbf{x}_{cm} = \frac{\int_V \rho(\mathbf{x})\mathbf{x}\, dV}{m}, \tag{4.25}$$

the different definitions of rigid bodies allow for different ways of representing a rigid body in a simulation. It may even depend on the specific application which representation is best used for the task at hand. What makes rigid body simulation significantly more complex than particle simulation is the fact that a rigid body can rotate around itself. It has three translational and three rotational degrees of freedom which need to be tracked in the simulation. Apart from tracking the rotation of a rigid body the question in what kind of coordinate system rotation and orientation of a rigid body should be represented needs to be discussed. Natural choices for coordinate systems would be the *local coordinate space* or *world coordinate space*.

4.2.1 Local Coordinate Space and World Coordinate Space

In the local coordinate space the rigid body's center of mass is equal to the origin of the local coordinate system and the principal axes of the rigid body are aligned with the x-y-z-axes. When the rigid body rotates, the axes of the local coordinate system change their orientation as well. Thus, the axes of the local coordinate system always stay aligned with the principal axes of the rigid body. The world coordinate system is fixed, it is equivalent to the coordinate system of the simulation domain and acts as the reference to which all globally defined coordinate information refers to. In a rigid body simulation calculations are done in both local and world coordinate space depending on which coordinate system is beneficial for the specific task. The orientation of a rigid body can be represented by a vector where each component of the vector stands for the angle of rotation around the respective coordinate axis. These angles around the coordinate axes are called *euler angles*. Euler angles suffer from a problem called

gimbal lock that can occur in a numerical simulation when two axes align which causes a loss of one degree of freedom [93]. More promising alternatives are rotation matrices or quaternions [79]. These rotation matrices not only store the current orientation, they are also used to transform the rigid body from local coordinates to world coordinates. A quaternion is a hyper-complex number that represents a rotation in three-dimensional space, it can also be converted to a rotation matrix and be used to transform local coordinates to world coordinates.

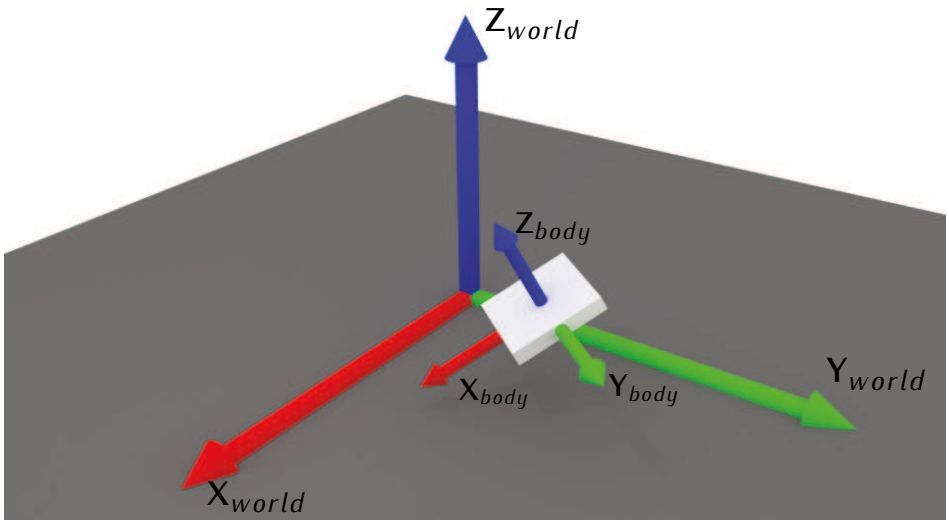


Figure 4.2: A rigid body in world space coordinates with its local coordinate axes

4.2.2 Moment of Inertia Tensor

Having established how rotations are represented we can turn to the concept of *inertia*. Inertia is the resistance of an object to change its state of motion. For translational motion the mass m describes the resistance to a change in velocity, the angular equivalent in 2D for mass is the mass moment of inertia:

$$L(t) = I\boldsymbol{\omega}(t) \tag{4.26}$$

$$I = mr^2, \tag{4.27}$$

here in the two-dimensional case I is a scalar. In the three-dimensional case a single scalar is not enough to fully describe the inertial behavior of a rigid body. For example an object that is long and thin shows different rotational behavior depending on what axis it rotates around. In general a three-dimensional object that is not symmetric with respect to its principal axes does not display the same rotational behavior for each axis. We will now briefly illustrate the derivation of how the concept of mass moment of inertia is established in 3D. We assume that a rigid body rotates around the origin of the coordinate system and we denote by $\mathbf{r}_i = (x_i, y_i, z_i)^T$ the position of the i-th

particle of our rigid body. If we sum up the individual angular momentum L_i for each particle, we obtain the angular momentum of the rigid body:

$$\begin{aligned}\mathbf{L} = \sum_i L_i &= \sum_i \mathbf{r}_i \times \mathbf{p}_i \\ &= \sum_i \mathbf{r}_i \times (m_i \mathbf{v}_i) \\ &= \sum_i m_i \mathbf{r}_i \times (\boldsymbol{\omega}_i \mathbf{v}_i) \\ &= \sum_i m_i \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \times \left(\begin{bmatrix} \omega_{x_i} \\ \omega_{y_i} \\ \omega_{z_i} \end{bmatrix} \times \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \right).\end{aligned}$$

From here we can expand the cross-products which yields:

$$\mathbf{L} = \sum_i m_i \begin{bmatrix} (y_i^2 + z_i^2)\omega_{x_i} - x_i y_i \omega_{y_i} - x_i z_i \omega_{z_i} \\ -y_i z_i \omega_{x_i} + (x_i^2 + z_i^2)\omega_{y_i} - y_i z_i \omega_{z_i} \\ -z_i x_i \omega_{x_i} - z_i y_i \omega_{y_i} + (x_i^2 + y_i^2)\omega_{z_i} \end{bmatrix}.$$

In the next step our aim is to write the vector $\mathbf{L}$ as the result of a matrix-vector multiplication which we can achieve by reordering the equation:

$$\begin{aligned}\mathbf{L} &= \begin{bmatrix} \sum_i m_i(y_i^2 + z_i^2) & -\sum_i m_i x_i y_i & -\sum_i m_i x_i z_i \\ -\sum_i m_i y_i z_i & +\sum_i m_i(x_i^2 + z_i^2) & -\sum_i m_i y_i z_i \\ -\sum_i m_i z_i x_i & -\sum_i m_i z_i y_i & +\sum_i m_i(x_i^2 + y_i^2) \end{bmatrix} \begin{bmatrix} \omega_{x_i} \\ \omega_{y_i} \\ \omega_{z_i} \end{bmatrix} \\ &= \begin{bmatrix} I_{xx} & -I_{yx} & -I_{zx} \\ -I_{xy} & I_{yy} & I_{zy} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \boldsymbol{\omega} = \mathbf{I}\boldsymbol{\omega}. \end{aligned} \tag{4.28}$$

The 3×3 matrix $\mathbf{I}$ in equation (4.28) is the 3D analogon to mass moment of inertia and is also called the *moment of inertia tensor*. The elements of the diagonal of $\mathbf{I}$ are the *moment of inertia coefficients*:

$$\begin{aligned} I_{xx} &= \sum_i m_i(y_i^2 + z_i^2) \\ I_{yy} &= \sum_i m_i(x_i^2 + z_i^2) \\ I_{zz} &= \sum_i m_i(x_i^2 + y_i^2), \end{aligned}$$

here I_{xx} represents the inertia for a rotation around the x-axis of the rigid body while I_{yy}, I_{zz} fulfill the same role for the y- and z-axis.

The off-diagonal entries are called the *products of inertia*:

$$I_{xy} = \sum_i m_i x_i y_i$$
$$I_{xz} = \sum_i m_i x_i z_i$$
$$I_{yz} = \sum_i m_i y_i z_i.$$

The products of inertia can be interpreted in a physical way as a measure of the imbalance of mass distribution of the rigid body. If the rigid body is perfectly symmetric with respect to its principal axes (i.e. a sphere), the products of inertia are zero and $\mathbf{I}$ takes the form of a diagonal matrix. Furthermore, we can observe from their definition that the moment of inertia tensor $\mathbf{I}$ is always symmetric and has positive entries on the diagonal. Knowledge of the products of inertia also allows us to define the term *principal axes* or *principal axes of inertia*. The principal axes of a rigid body are the three mutually orthogonal axes of a coordinate system for which the products of inertia become zero. A method how such axes can be obtained for any rigid body using the *Principal Axis Transformation* is described by Strang [82].

So far we have shown the formulas for the moment of inertia tensor for rigid bodies that are composed of discrete particles. When we consider rigid bodies as a continuous volume, the discrete sum is replaced by an integral of the density distribution $\rho(\mathbf{x})$ over the volume of the rigid body:

$$\begin{aligned} I_{xx} &= \int_V \rho(\mathbf{x})(y_i^2 + z_i^2)dV \\ I_{yy} &= \int_V \rho(\mathbf{x})(x_i^2 + z_i^2)dV \\ I_{zz} &= \int_V \rho(\mathbf{x})(x_i^2 + y_i^2)dV \\ I_{xy} &= \int_V \rho(\mathbf{x})x_i y_i dV \\ I_{xz} &= \int_V \rho(\mathbf{x})x_i z_i dV \\ I_{yz} &= \int_V \rho(\mathbf{x})y_i z_i dV. \end{aligned} \tag{4.29}$$

Another property of the moment of inertia tensor is that it is always dependent on a specific center of rotation. So if we calculate $\mathbf{I}$ with respect to the center of mass of the rigid body, the tensor will remain constant in local coordinates, regardless of the position of the body in the world coordinate system of the simulation domain. In a rigid body simulation it may be necessary to use the moment of inertia tensor in world coordinates. In order to avoid recomputing the tensor every time it is needed in

a specific calculation the tensor can be transformed into the world coordinate system by a similarity transformation using the orientation information stored in the rotation matrix $\mathbf{R}(t)$:

$$\mathbf{I}_w(t) = \mathbf{R}(t)\mathbf{I}\mathbf{R}(t)^T \tag{4.30}$$

We can now discuss how we can obtain the center of mass and the moment of inertia tensor of a rigid body in practice. For basic rigid body geometries like spheres, boxes or cylinders the center of mass is the geometric center of the object and the moment of inertia tensor in local coordinates can be obtained from literature [34, 46]. If we assume the presence of a finite element framework alongside of our rigid body simulator, as is case in our plan, we can use it to evaluate the volume integrals in equations (4.25, 4.29). At first we need to find the center of mass of the rigid body, then we translate the rigid body to the origin of the coordinate system so that the local coordinate system and the world coordinate system coincide. In this configuration we can use the FEM-framework to calculate the integrals for the moment of inertia tensor, the exact procedure is described in algorithm 6.

Algorithm 6: Center of Mass and Moment of Inertia Tensor

```
Data: obj, mesh
Result: com, moi,vol
begin
    com = (0,0,0)
    vol = 0
    for i = 1 to mesh.nel do
        PointClassification(mesh, i, obj)
    for i = 1 to mesh.nel do
        if obj.PointInObject(mesh.elements[i])==solid then
            evalComEquation(i,com)
            vol+=mesh.getElementVolume(i)
    for i = 1 to mesh.nel do
        if obj.PointInObject(mesh.elements[i]) == solid then
            evalMoiTensorEquation(i, com, moi)
```

4.2.3 Equations of Motion for Rigid Bodies

A main component of a rigid body simulator is a solver for the rigid body equations of motion. To update the position and the orientation of a rigid body in our simulation we

need to solve a system called the *Newton-Euler equations*:

$$\begin{aligned}\dot{\mathbf{r}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= \frac{\mathbf{F}}{m} \\ \dot{\mathbf{q}} &= \mathbf{Q}\boldsymbol{\omega} \\ \dot{\boldsymbol{\omega}} &= \mathbf{I}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega})),\end{aligned} \tag{4.31}$$

here $\boldsymbol{\tau}$ is the time-derivative of the angular momentum:

$$\boldsymbol{\tau}(t) = \frac{d\mathbf{L}(t)}{dt} = \frac{d\mathbf{I}(t)\boldsymbol{\omega}(t)}{dt} = \frac{d\mathbf{I}(t)}{dt}\boldsymbol{\omega}(t) + \mathbf{I}(t)\frac{d\boldsymbol{\omega}(t)}{dt}, \tag{4.32}$$

the time-derivative $\frac{d\mathbf{I}(t)}{dt}\boldsymbol{\omega}(t)$ is complicated to compute and we refer to Eberly [22] for a derivation. With these results we obtain:

$$\boldsymbol{\tau}(t) = \boldsymbol{\omega}(t) \times (\mathbf{I}(t)\boldsymbol{\omega}(t)) + \mathbf{I}(t)\frac{d\boldsymbol{\omega}(t)}{dt} \tag{4.33}$$

4.2.4 Friction

The surface of a rigid body cannot always be considered as perfectly smooth. When two perfectly smooth rigid bodies touch and move in parallel directions to each other there will not be any force between those rigid bodies because the smoothness of their surfaces ensures a perfect glide. In reality, as we pointed out before, the assumption of perfectly smooth surfaces cannot generally be made. Surfaces generally have a certain degree of roughness to them. If two such surfaces were in contact and moving parallel to each other a force would arise because of the roughness of the surfaces that resists to the parallel movement of the rigid bodies. This force is called *friction*.

The friction force is modeled by Coulomb's law of friction. In this model the friction force is dependent on the external forces (i.e. gravity) that initiate the contact between the two rigid bodies' surfaces and the *coefficient of friction* μ which is a measure for the roughness of the surface of a rigid body. The coefficient of friction is usually determined by experimental measurements. The prototypical situation of a frictional contact is a ball rolling down an inclined plane. The external force of gravity is pulling the ball straight down towards the earth core. A *normal force* $\mathbf{F}_n$ is acting in the normal direction of the inclined plane in order to prevent the ball from penetrating the plane. The friction force is acting in a direction parallel to the surface of the plane against the downward motion of the ball. An illustration of these forces is shown in the free body diagram in figure 4.3.
In the Coulomb model a distinction is made between *static friction* and *dynamic friction*.

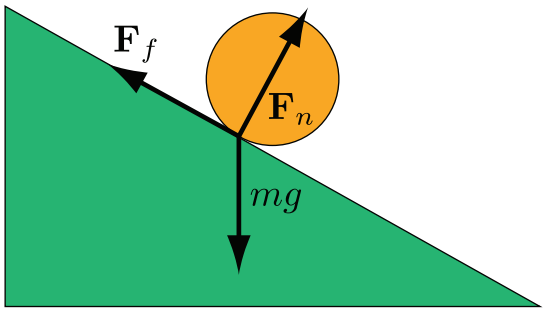


Figure 4.3: Illustration of the forces acting on a ball rolling on an inclined plane

Static Friction

In the situation that a ball is resting on a plane surface and a force is applied that wants to move the ball in a tangential direction to the plane there is also a force that is opposing the tangential force. This force is the resistance to initiate movement and is called the static friction $\mathbf{F}_s$. The maximum static friction is

$$\|\mathbf{F}_{s,max}\| = \mu_s \|\mathbf{F}_n\|, \tag{4.34}$$

here μ_s is the coefficient of static friction. This relation tells us that as long as the applied force is below $\|\mathbf{F}_{s,max}\|$ the body will not start to move. When the applied force grows larger than $\|\mathbf{F}_{s,max}\|$ movement is initiated and the friction regime changes from static friction to dynamic friction.

Dynamic Friction

In the case that a tangential motion between two rigid bodies is initiated we are in the dynamic friction regime. The direction of the dynamic friction force is in the opposite direction as the tangential movement as was the case for static friction. The magnitude of the dynamic friction is characterized by the normal force and the *coefficient of dynamic friction* μ_d:

$$\|\mathbf{F}_d\| = \mu_d \|\mathbf{F}_n\|. \tag{4.35}$$

Typically, the coefficient of dynamic friction is smaller than the coefficient of static friction which is understandable by intuition that a body that is already moving is showing less resistance than a body that has to be transferred from resting state to moving state.

Friction in 3D

In the three-dimensional case a frictional contact is described by a contact plane and a normal force vector similarly to the 2D case (see figure 4.4). The contact plane itself is defined by two tangential vectors $\mathbf{t}_u$ and $\mathbf{t}_v$. The friction force then lies in the contact plane spanned by $\mathbf{t}_u$ and $\mathbf{t}_v$. The friction force $\mathbf{F}_{friction}$ can be written as a

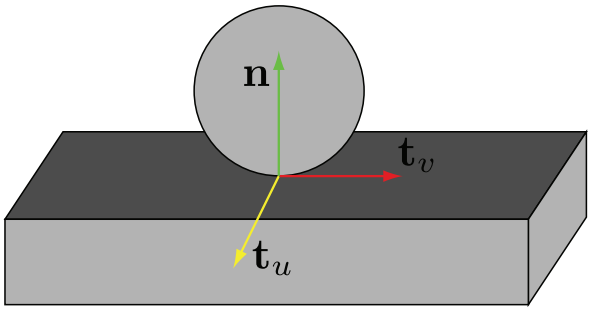


Figure 4.4: A frictional contact in 3D

linear combination of the tangential vectors $\mathbf{F}_{friction} = \mathbf{t}_u f_u + \mathbf{t}_v f_v$, where f_u and f_v are the magnitudes of the tangential forces. In 3D the Coulomb friction model can be formulated as:

$$f_x^2 + f_y^2 \leq \mu^2 f_n^2. \tag{4.36}$$

The geometrical interpretation of equation (4.36) is a cone with the apex at the point of contact and base in the direction of the normal force as shown in figure 4.5. The Coulomb model also states that the sum of the friction force $F_{friction}$ and the normal force lies in the friction cone $\mathbf{F} = \mathbf{F}_n + \mathbf{F}_u + \mathbf{F}_v$.

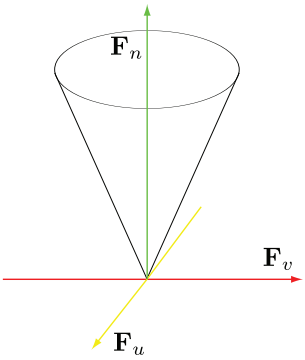


Figure 4.5: Illustration of the forces acting on a ball on an inclined plane

4.3 Single Body Collision Model

In the discussion of our simulator modules we arrived at the point that a set of contact points, a set of collision normals and possibly distance information was calculated, the union of these quantities we called the contact set or reduced contact set. In this section we will resume at this point and introduce a model that is capable of describing contact configurations and calculating contact forces that resolve a collision between one single pair of rigid bodies, meaning that the contact forces will prevent non-physical penetration of rigid bodies by changing the velocity accordingly. In figure 4.6 we see

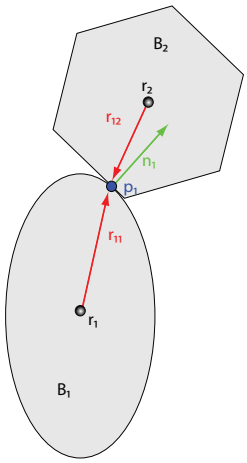


Figure 4.6: In the figure a prototypical collision between two rigid bodies $\mathbf{B}_1, \mathbf{B}_2$ with contact point $\mathbf{p}_1$ and contact normal $\mathbf{n}_1$. The vectors $\mathbf{r}_{11}, \mathbf{r}_{12}$ point from the center of mass to the contact point.

a collision between two rigid bodies and the associated contact set. The first term that we need to look at in the context of computing collision forces is the relative velocity of two rigid bodies A, B, velocities $\mathbf{v}_A$, $\mathbf{v}_B$ and angular velocities $\boldsymbol{\omega}_A$, $\boldsymbol{\omega}_B$:

$$\mathbf{v}_{AB} = (\mathbf{v}_A + \boldsymbol{\omega}_A \times \mathbf{r}_A - (\mathbf{v}_B + \boldsymbol{\omega}_B \times \mathbf{r}_B)). \tag{4.37}$$

Here $\mathbf{r}_A$, $\mathbf{r}_B$ are vectors from the center of the respective rigid body to the contact point. From the relative velocity we can compute the relative normal velocity that allows us to classify the collision state of the rigid bodies:

$$\mathbf{n} \cdot \mathbf{v}_{AB} = \mathbf{v}_{AB,n}.$$

Based on the value of the relative velocity we can see whether a pair of rigid bodies is on a collision course, in a resting state or in a state where the distance along the normal direction increases and the bodies separate:

$$\begin{aligned} \mathbf{n} \cdot \mathbf{v}_{AB} &< 0 : \text{colliding} \\ \mathbf{n} \cdot \mathbf{v}_{AB} &= 0 : \text{touching} \\ \mathbf{n} \cdot \mathbf{v}_{AB} &> 0 : \text{separating}. \end{aligned}$$

The relative velocity in normal direction tells us the change of the distance in normal direction is going to decrease (< 0), remain the same ($= 0$) or increase (> 0) (see figure 4.7). The idea is that for a pair of rigid bodies that has been reported by the collision detection system as sufficiently close and where the relative normal velocity is smaller than zero to change the velocity and the angular velocity of the rigid bodies

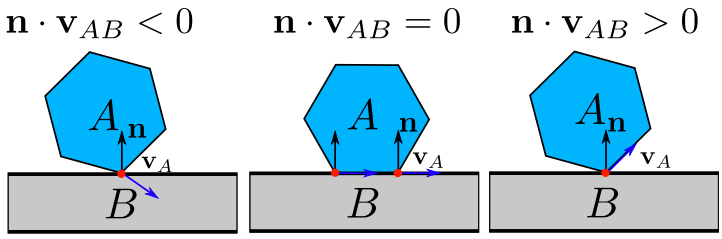


Figure 4.7: Based on the sign of the relative normal velocity we can determine the contact configuration of a pair of rigid bodies.

by an impulse:

$$\mathbf{v}_A(t + \Delta t) = \mathbf{v}_A(t) + \frac{f\mathbf{n}}{m_A} \tag{4.38}$$

$$\boldsymbol{\omega}_A(t + \Delta t) = \boldsymbol{\omega}_A(t) + \mathbf{I}_A^{-1}(\mathbf{r}_A \times f\mathbf{n}_A) \tag{4.39}$$

$$\mathbf{v}_B(t + \Delta t) = \mathbf{v}_B(t) - \frac{f\mathbf{n}}{m_B} \tag{4.40}$$

$$\boldsymbol{\omega}_B(t + \Delta t) = \boldsymbol{\omega}_B(t) + \mathbf{I}_B^{-1}(\mathbf{r}_B \times f\mathbf{n}_B). \tag{4.41}$$

Here we compute the so called post-impulse velocities $\mathbf{v}_A(t+\Delta t), \mathbf{v}_B(t+\Delta t)$ as well as the corresponding post-impulse angular velocities of our rigid bodies in the next time step by applying an impulse $\frac{f\mathbf{n}}{m}$ to them that will cause the pre-impulse velocity to change in a way that a physical non-penetration collision state in the next time step is achieved. The pre-impulse and post-impulse velocities are often written as $\mathbf{v}_-$ and $\mathbf{v}_+$. The magnitude f of the impulse can be computed as:

$$f = \frac{-(1+\epsilon)(\mathbf{n}_A(\mathbf{v}_A - \mathbf{v}_B) + \boldsymbol{\omega}_A(\mathbf{r}_A \times \mathbf{n}_A) - \boldsymbol{\omega}_B(\mathbf{r}_B \times \mathbf{n}_B))}{m_A^{-1} + m_B^1 + (\mathbf{r}_A \times \mathbf{n}_A)^T \mathbf{I}_A^{-1}(\mathbf{r}_A \times \mathbf{n}_A) + (\mathbf{r}_B \times \mathbf{n}_B)^T \mathbf{I}_B^{-1}(\mathbf{r}_B \times \mathbf{n}_B)} \tag{4.42}$$

The value $\epsilon \in [0, 1]$ is the coefficient of restitution which is used to model a possible loss of kinetic energy that can occur during collision. A perfectly elastic collision (all kinetic energy is conserved) corresponds to $\epsilon = 1$ and a perfectly inelastic collision (all kinetic energy in normal direction is lost) corresponds to $\epsilon = 0$.

4.4 Multi-Body Collision Models

4.4.1 Introduction

In particulate flow simulations multiple collisions need to be handled at the same time. The single-body collision model that we have introduced in the previous section is the basis for deriving models that are able to handle multiple particle collisions at the same time. Another interpretation of multiple collisions at the same time is handling multiple contact points at the same time where the contact force f_0 at a contact point

c_0 is influenced by the contact force f_1 at another contact point c_0 (see figure 4.8). Different approaches exist for dealing with multiple collisions and the following are

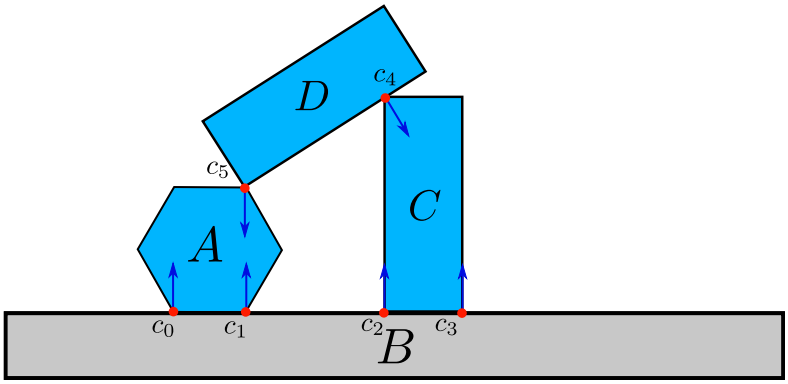


Figure 4.8: A collision situation with multiple contact points. Four colliding bodies with different contact points are shown. From intuition it should become clear that the contact force value in contact points c_0, c_1 is dependent on how much force body D exerts onto body A by pushing on it in contact point c_5. A similar situation exists for contact points c_2, c_3 and c_4.

used in this work:

1. Constrained-based methods

2. Impulse- or Sequential Impulse-based methods

3. DEM-based methods

In constrained-based rigid body simulation we formulate a set of constraints that our rigid bodies have to fulfill in order to achieve physical behavior of our rigid bodies. The constrained-based methods can be divided into acceleration-based methods and velocity-based methods. The difference between these methods is the physical quantity according to which the mathematical formulation of the constraint is done. The downside of acceleration-based formulations is that they suffer from indeterminacy, inconsistency [81, 4] and the so called small time step problem as has been pointed out by Milenkovic and Schmidl [75, 74]. Approaches that operate on the velocity level instead of the acceleration level do not suffer from these problems [25]. In a velocity-based formulation the effect of a force is seen over an interval equivalent to the time step size. So, if we knew the true contact force $\mathbf{f}_t(t)$ then we can write the impulse $\mathbf{J}$ as the integral:

$$\mathbf{J} = \int_0^{\Delta t} \mathbf{f}_t(t) dt.$$

We can rewrite this as

$$\int_0^{\Delta t} m\frac{d\mathbf{v}}{d}dt = \int_0^{\Delta t} \mathbf{f}_t(t)dt$$
$$m(\mathbf{v}^{\Delta t} - \mathbf{v}^0) = \mathbf{J}.$$

This leads to the impulse $\mathbf{J}$ which can be applied to yield the new velocity. The force in the velocity-based formulation can be obtained by

$$\mathbf{J} = \Delta t \mathbf{f}.$$

In order to find the new position we can then just apply an integration scheme with the updated velocity, which will lead to the same result as if we knew the true force $\mathbf{f}_t$ and evaluated the time integral over the time step and from there on continued to solve for the new velocity and position.

Impulse-based methods are used and described in the works of Mirtich [56] or Guendelman [37]. Sequential impulse methods solve the problem of multiple contact points by iterating the force calculation in a single time step of the simulation until the relative normal velocities in all contact points is such that no non-physical interpenetrations can appear in the next time step. The difference between sequential impulse and constraint-based methods is subtle, it is also possible to call sequential impulse methods a different type of a constraint solver. When the impulses computed in a sequential impulse solver aim to satisfy for example a non-penetration constraint, then this is indeed just a different constraint solver. Instead of assembling a system of constraints and then solving them simultaneously in sequential impulses the constraints are solved sequentially and the procedure is iterated until the changes in the impulse values between successive iterations become small enough to characterize the procedure as converged. In DEM-based methods a soft sphere collision model is evaluated for each pair of colliding bodies in each time step [50]. In DEM-simulations the time step size Δt is usually very small in order to propagate collision forces occurring from multiple particle collisions.

4.4.2 Velocity-based Multi-Body Collision

For our particulate flow solver we opted to explore other possibilities than the traditional 'short range repulsive forces' [88] that are found in many particulate flow codes for collision treatment. The traditional short range repulsive force model in its standard form has no handling for multiple contact points on one rigid body and some configurations like stable stacking are hard to achieve with a model that applies forces based on distance only. So one of our choices to avoid such limitations is the model from rigid body simulation that is based on the works of Baraff [7], Sauer and Schoemer [73]. The advantage of the model is that it can handle multiple contact points and arbitrary rigid bodies, it is also well suited to support geometries that are defined by

surface triangulations, analytic descriptions or even level-set representations. At first we will formulate the Newton-Euler equations of motions for a system of rigid bodies as it was proposed in the model of the before-mentioned authors. For a system of n rigid bodies with K contact points we arrive at the following system of equations:

$$\dot{\mathbf{r}}_i = \mathbf{v}_i \tag{4.43}$$

$$\dot{\mathbf{q}}_i = \frac{1}{2}\boldsymbol{\omega}_i \mathbf{q}_i \tag{4.44}$$

$$\dot{\mathbf{v}}_i = m_i^{-1} \sum_{j_k=i} \mathbf{f}_k - m_i^{-1} \sum_{i_k=i} \mathbf{f}_k + m_i^{-1} \mathbf{f}_i^{ext} \tag{4.45}$$

$$\dot{\boldsymbol{\omega}}_i = \mathbf{I}_i^{-1} \sum_{j_k=i} \mathbf{r}_{kj} \times \mathbf{f}_k - \mathbf{I}_i^{-1} \sum_{i_k=i} \mathbf{r}_{ki} \times \mathbf{f}_k - \mathbf{I}_i^{-1} \boldsymbol{\omega}_i \times \mathbf{I}_i \boldsymbol{\omega}_i + \mathbf{I}_i^{-1} \boldsymbol{\tau}_i^{ext}. \tag{4.46}$$

Where the vectors $\mathbf{v}_i$ are the linear velocity of the rigid bodies, $\mathbf{q}_i$ the orientation of the rigid bodies, $\boldsymbol{\omega_i}$ the angular velocities, $\mathbf{f}_k$ the contact forces in the k-th contact point, $\mathbf{v}_i^{ext}$ and $\boldsymbol{\tau}_i^{ext}$ the sum of external forces, respectively external torques acting on our rigid bodies. The external forces and torques in this formulation of the Newton-Euler equations are where we can insert the hydrodynamic force and torque that the fluid in our CFD-simulation exerts on the rigid bodies.

In Matrix-Vector form we can write the system as:

$$\dot{\mathbf{s}} = \mathbf{S}\mathbf{u} \tag{4.47}$$

$$\dot{\mathbf{u}} = \mathbf{M}^{-1}(\mathbf{C}\mathbf{N}\mathbf{f} + \mathbf{f}^{ext}). \tag{4.48}$$

Here $\mathbf{s} \in \mathbb{R}^{7n}$ is the vector which describes both position and orientation of the rigid bodies at the same time:

$$\mathbf{s} = \left[\mathbf{r}_1, \mathbf{q}_1, ..., \mathbf{r}_n, \mathbf{q}_n\right]^T. \tag{4.49}$$

Furthermore, $\mathbf{u} \in \mathbb{R}^{6n}$ is the vector that groups the linear velocity and the angular velocity:

$$\mathbf{u} = \left[\mathbf{v}_1, \boldsymbol{\omega}_1, ..., \mathbf{v}_n, \boldsymbol{\omega}_n\right]^T. \tag{4.50}$$

The vector $\mathbf{f} \in \mathbb{R}^k$ has the contact forces as its components:

$$\mathbf{f} = \left[\mathbf{f}_1, ..., \mathbf{f}_K\right]^T. \tag{4.51}$$

The external forces and external torques $\mathbf{f}^{ext} \in \mathbb{R}^{6n}$ are grouped in the vector:

$$\mathbf{f}^{ext} = \left[\mathbf{f}_1^{ext}, \boldsymbol{\tau}_1^{ext} - \boldsymbol{\omega}_1 \times \mathbf{I}_1 \boldsymbol{\omega}_1, ..., \mathbf{f}_n^{ext}, \boldsymbol{\tau}_n^{ext} - \boldsymbol{\omega}_n \times \mathbf{I}_n \boldsymbol{\omega}_n\right]^T. \tag{4.52}$$

Since we represent orientations as quaternions we need a way to update our orientation in matrix form in order to write this update in matrix vector form. So the multiplication

of $\frac{1}{2}\boldsymbol{\omega}_i\mathbf{q}_i$ can be written as $\mathbf{Q}_i\boldsymbol{\omega}_i$ with $\mathbf{q}_i = [s_i, x_i, y_i, z_i] \in \mathbb{R}^4$ and $\mathbf{Q}_i$:

$$\mathbf{Q}_i = \frac{1}{2}\begin{bmatrix} -x_i & -y_i & -z_i \\ s_i & z_i & -y_i \\ -z_i & s_i & x_i \\ y_i & -x_i & s_i \end{bmatrix}. \tag{4.53}$$

We can group the $\mathbf{Q}_i$ in another matrix $\mathbf{S} \in \mathbb{R}^{7n\times 6n}$:

$$\mathbf{S} = \begin{bmatrix} \mathbf{1}_{3\times 3} & & & & 0 \\ & \mathbf{Q}_1 & & & \\ & & \ddots & & \\ & & & \mathbf{1}_{3\times 3} & \\ 0 & & & & \mathbf{Q}_n \end{bmatrix}. \tag{4.54}$$

Moreover, $\mathbf{M} \in \mathbb{R}^{6n\times 6n}$ is the matrix that has the rigid body masses as entries:

$$\mathbf{M} = \begin{bmatrix} m_i\mathbf{1}_{3\times 3} & & & & 0 \\ & \mathbf{I}_1 & & & \\ & & \ddots & & \\ & & & m_n\mathbf{1}_{3\times 3} & \\ 0 & & & & \mathbf{I}_n \end{bmatrix}. \tag{4.55}$$

The K contact normals can be represented as a matrix $\mathbf{N} \in \mathbb{R}^{3K\times K}$:

$$\mathbf{N} = \begin{bmatrix} \mathbf{n}_1 & & 0 \\ & \ddots & \\ 0 & & \mathbf{n}_K \end{bmatrix}. \tag{4.56}$$

The matrix $\mathbf{C} \in \mathbb{R}^{6n\times 3K}$ a matrix that represents a contact condition:

$$\mathbf{C}_{lk} = \begin{cases} -\mathbf{1}_{3\times 3} & \text{for } l = 2_{i_k} - 1 \\ -\mathbf{r}^{\times}_{ki_k} & \text{for } l = 2_{i_k} \\ \mathbf{1}_{3\times 3} & \text{for } l = 2_{j_k} - 1 \\ \mathbf{r}^{\times}_{kj_k} & \text{for } l = 2_{j_k} \\ \mathbf{0} & \text{else} \end{cases}. \tag{4.57}$$

Where the matrix $\mathbf{r}^{\times} \in \mathbb{R}^{3\times 3}$ with $\mathbf{r}^{\times}\mathbf{a} = \mathbf{r} \times \mathbf{a}$ for $\mathbf{a}, \mathbf{r} \in \mathbb{R}^3$ is

$$\mathbf{r}^{\times} = \begin{bmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{bmatrix}. \tag{4.58}$$

The Newton-Euler equations, discretized by an explicit Euler scheme, can be written in matrix-vector form as:

$$\mathbf{s}^{t+\Delta t} = \mathbf{s}^t + \Delta t \mathbf{S}\mathbf{u}^{t+\Delta t}, \tag{4.59}$$
$$\mathbf{u}^{t+\Delta t} = \mathbf{u}^t \Delta t \mathbf{M}^{-1}(\mathbf{C}\mathbf{N}\mathbf{f}^{t+\Delta t} + \mathbf{f}^{ext}). \tag{4.60}$$

Special attention we would like to draw to the matrix $\mathbf{C}$. This matrix has as many columns as contact points, the number of rows is equal to the number of rigid bodies. By multiplication with the matrix $\mathbf{P}_k^T \in \mathbb{R}^{3K\times 3}$

$$\mathbf{P}_k^T = \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \ldots, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \ldots, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \tag{4.61}$$

the k-th contact condition can be extracted from the matrix $\mathbf{C}$.

$$\mathbf{n}_k^T \mathbf{P}_k^T \mathbf{C}^T \mathbf{u} = \mathbf{n}_k^T(\mathbf{v}_{j_k} + \boldsymbol{\omega}_{j_k} \times \mathbf{r}_{kj_k}) - \mathbf{n}_k^T(\mathbf{v}_{i_k} + \boldsymbol{\omega}_{i_k} \times \mathbf{r}_{ki_k}) \tag{4.62}$$

If the rigid bodies $\mathbf{B}_{ik}$ und $\mathbf{B}_{jk}$ are touching in a contact point $\mathbf{p}_k$ the following condition is fulfilled:

$$\mathbf{n}_k^T \mathbf{P}_k^T \mathbf{C}^T \mathbf{u}^{t+\Delta t} \geq 0 \text{ complementary to } \mathbf{f}_k \geq 0 \tag{4.63}$$

This condition is called the complementarity condition. The meaning of complementarity is here that either there is no colliding contact ($\mathbf{n}_k^T \mathbf{P}_k^T \mathbf{C}^T \mathbf{u}^{t+\Delta t} > 0$) and because of this the contact forces are zero ($f_k = 0$) or there is a contact ($\mathbf{n}_k^T \mathbf{P}_k^T \mathbf{C}^T \mathbf{u}^{t+\Delta t} = 0$) and hence there are contact forces active ($f_k > 0$).
The complementarity condition was formulated by Sauer and Schoemer [73] in such a way that potentially colliding, meaning very close, contacts are considered, which is advantageous when larger time step sizes are used. In this form the complementarity condition is:

$$\mathbf{n}_k^T \mathbf{P}_k^T \mathbf{C}^T \mathbf{u}^{t+\Delta t} \geq \frac{\nu_k}{\Delta t} \text{ complementary to } f_k \geq 0. \tag{4.64}$$

The detailed calculation of the ν_k is described in the works of Sauer and Schoemer [73]. Let $\nu = [\nu_1, ..., \nu_n]^T \in \mathbb{R}^K$ and $\nu_k = 0$ for all touching contacts then a complementarity condition for potential and touching contacts can be formulated as:

$$\mathbf{N}^T \mathbf{C}^T \mathbf{u}^{t+\Delta t} \geq \frac{\nu}{\Delta t} \text{ complementary to } \mathbf{f} \geq 0 \tag{4.65}$$

By substituting eq. (4.60) into eq. (4.65) we get:

$$\mathbf{N}^T \mathbf{C}^T (\mathbf{u}^t + \Delta t \mathbf{M}^{-1}(\mathbf{C}\mathbf{N}\mathbf{f}^{t+\Delta t} + \mathbf{f}^{ext})) - \frac{\nu}{\Delta t} \geq \mathbf{0}. \tag{4.66}$$

By reordering the terms we arrive at:

$$\underbrace{\mathbf{N}^T \mathbf{C}^T \mathbf{M}^{-1} \mathbf{C}\mathbf{N}}_{\mathbf{A}} \underbrace{\Delta t \mathbf{f}^{t+\Delta t}}_{\mathbf{x}} + \underbrace{\mathbf{N}^T \mathbf{C}^T (\mathbf{u}^t + \Delta t \mathbf{M}^{-1} + \mathbf{f}^{ext}) - \frac{\nu}{\Delta t}}_{\mathbf{b}} \geq \mathbf{0}. \tag{4.67}$$

So the final problem is of the form:

$$\mathbf{A}\mathbf{x} + \mathbf{b} \geq \mathbf{0} \text{ complementary to } \mathbf{x} \geq 0 \tag{4.68}$$

with $\mathbf{A} \in \mathbb{R}^{K \times K}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^K$. This is the well known linear complementarity problem (LCP) [18]. The entries of the matrix $\mathbf{A}$ are calculated as follows:

$$\begin{aligned}
\mathbf{A}_{lk} &= \delta_{i_l i_k} \mathbf{n}_l^T (\frac{1}{m_{i_k}} \mathbf{1}_{3\times 3} - \mathbf{r}_{li_l}^{\times} \mathbf{I}_{i_k}^{-1} \mathbf{r}_{ki_k}^{\times}) \mathbf{n}_k && (4.69)\\
&- \delta_{i_l j_k} \mathbf{n}_l^T (\frac{1}{m_{j_k}} \mathbf{1}_{3\times 3} - \mathbf{r}_{li_l}^{\times} \mathbf{I}_{j_k}^{-1} \mathbf{r}_{kj_k}^{\times}) \mathbf{n}_k && (4.70)\\
&- \delta_{j_l i_k} \mathbf{n}_l^T (\frac{1}{m_{i_k}} \mathbf{1}_{3\times 3} - \mathbf{r}_{lj_l}^{\times} \mathbf{I}_{i_k}^{-1} \mathbf{r}_{ki_k}^{\times}) \mathbf{n}_k && (4.71)\\
&+ \delta_{j_l j_k} \mathbf{n}_l^T (\frac{1}{m_{j_k}} \mathbf{1}_{3\times 3} - \mathbf{r}_{lj_l}^{\times} \mathbf{I}_{j_k}^{-1} \mathbf{r}_{kj_k}^{\times}) \mathbf{n}_k, && (4.72)
\end{aligned}$$

here δ is the Kroneckerdelta. The system matrix $\mathbf{A}$ which has been built in the described form is positive-definite [25] and the associated LCP can be solved with efficient iterative methods like the projected Gauss-Seidel (PGS) or projected conjugate gradients [18, 25]. In order to formulate the contact problem as a LCP including friction, equation 4.69 has to be extended. The formulation here is based on the formulations described in the work of Erleben [25] and Sauer and Schoemer [73]. This standard formulation of the LCP for frictional contacts can described as:

$$\begin{bmatrix} \mathbf{D}^T\mathbf{C}^T\mathbf{M}^{-1}\mathbf{C}\mathbf{D} & \mathbf{D}^T\mathbf{C}^T\mathbf{M}^{-1}\mathbf{C}\mathbf{N} & \mathbf{E} \\ \mathbf{N}^T\mathbf{C}^T\mathbf{M}^{-1}\mathbf{C}\mathbf{D} & \mathbf{N}^T\mathbf{C}^T\mathbf{M}^{-1}\mathbf{C}\mathbf{N} & \mathbf{0} \\ -\mathbf{E}^T & \mu & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \Delta t\beta \\ \Delta t\mathbf{f} \\ \lambda \end{bmatrix} + \begin{bmatrix} \mathbf{D}^T\mathbf{C}^T\left(\mathbf{u}^t + \Delta t\mathbf{M}^{-1}\mathbf{f}_{ext}\right) \\ \mathbf{N}^T\mathbf{C}^T\left(\mathbf{u}^t + \Delta t\mathbf{M}^{-1}\mathbf{f}_{ext}\right) - \frac{\nu}{\Delta t} \\ \mathbf{0} \end{bmatrix}$$
$$\geq \mathbf{0}$$
$$\text{complementary to } \begin{bmatrix} \Delta t\beta \\ \Delta t\mathbf{f} \\ \lambda \end{bmatrix} \geq \mathbf{0}$$

Here $\mathbf{D}$ is a matrix representing the geometrical configuration of the friction cone approximation for the colliding bodies, $\mathbf{M}$ is the mass matrix of the bodies, $\mathbf{C}$ the matrix of contact conditions, $\mathbf{N}$ the matrix of normal vectors for the contacts. The problem is then solved for $\mathbf{f}$ and β which are vectors containing as components the magnitude of the normal and frictional impulses. Alternatively, a similar non-linear complementarity problem formulation (NCP) as described by Slicowitz, Niebe and Erleben [80] can be used which can be solved using a projected Gauss-Seidel (PGS) solver. Apart from the complementarity problem based formulations, we can use a solution technique for the contact force problem called sequential impulses (SI) as described by Guendelman [37]. Another model that is implemented in our code is a model based on the work of Harada [39] in which rigid bodies are modeled by a particle method. Contact forces are then calculated using a discrete element method (DEM) approach.

4.4.3 Linear Complementarity Problems

After having formulated the contact force problem as a linear complementarity problem we want to present the basic theory and a way of solving LCPs. For a detailed discussion of LCPs we refer to the work of Cottle, Pang and Stone [18]. The LCP is defined by a given matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and a vector $\mathbf{b} \in \mathbb{R}^n$. We should then find vectors $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$ such that

$$\mathbf{w} = \mathbf{A}\mathbf{x} - \mathbf{b} \tag{4.73}$$

$$\mathbf{x} \geq \mathbf{0} \tag{4.74}$$

$$\mathbf{w} \geq \mathbf{0} \tag{4.75}$$

where $\mathbf{x}$ and $\mathbf{w}$ satisfy the *complementarity condition*.

$$\mathbf{x}^T\mathbf{w} = \mathbf{0} \tag{4.76}$$

The task of finding a solution to an LCP is an NP-hard problem, but the works of Cottle et al. or Murty [18, 59] include solvers that have expected polynomial complexity. These solvers can be classified as direct or pivoting solvers or as iterative methods. In our work we will resort to iterative matrix solvers for the LCP. The general form for an iterative solver is based on splitting methods. An example for a direct LCP solver would be Lemke's Algorithm [49, 4]. A problem with Lemke's algorithm is though that in the basic implementation the method can terminate without a result which makes fall-back solver necessary.
Iterative solvers for linear equations such as the Jacobi method, Gauss-Seidel method, successive over relaxation method or conjugate gradient method [77] can be modified to solve LCPs as shown by Erleben [25] or Catto [14]. The advantage of iterative solvers is that they can always return an approximate solution and they can use results from previous time steps as an initial solution which has been shown to increase their performance [25]. We will now briefly summarize how to extend the classic Jacobi type iterative solvers to solve LCPs. Iterative methods usually rely on splitting the matrix $\mathbf{A}$ of linear equations. To arrive at a general equation for an iterative matrix solver we decompose the system matrix $\mathbf{A}$ into a strict lower triangular matrix $\mathbf{L}$, a diagonal matrix $\mathbf{D}$, and a strict upper triangular matrix $\mathbf{U}$:

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}. \tag{4.77}$$

The general problem of a system of linear equations is:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{4.78}$$

we can now substitute equation 4.77 into 4.78 to arrive at:

$$\begin{aligned} (\mathbf{L} + \mathbf{D} + \mathbf{U})\mathbf{x} &= \mathbf{b} \\ \mathbf{D}\mathbf{x} &= \mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x} \\ \mathbf{x} &= \mathbf{D}^{-1}\mathbf{b} - \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}. \end{aligned} \tag{4.79}$$

In order to compute the $i-$th component of the solution vector, we can write:

$$x_i = \frac{b_i - \sum_{j=0}^{i-1} \mathbf{L}_{i,j} x_j - \sum_{j=i+1}^{n-1} \mathbf{U}_{i,j} x_j}{\mathbf{A}_{i,i}}, \tag{4.80}$$

from this we can derive the iterative scheme:

$$x_i^{k+1} = \frac{b_i - \sum_{j=0}^{i-1} \mathbf{L}_{i,j} x_j^{k+1} - \sum_{j=i+1}^{n-1} \mathbf{U}_{i,j} x_j^{k+1}}{\mathbf{A}_{i,i}}. \tag{4.81}$$

The iterative scheme in equation (4.81) is the Jacobi method, moreover Murty [59] shows that a general iteration scheme for jacobi type solvers is given by:

$$\mathbf{x}^{k+1} = \lambda(\mathbf{x}^k - \omega \mathbf{E}^k(\mathbf{A}\mathbf{x}^k - \mathbf{b} + \mathbf{K}^k(\mathbf{x}^{k+1} - \mathbf{x}^k))) + (1-\lambda)\mathbf{x}^k. \tag{4.82}$$

To arrive at the Jacobi method we choose $\mathbf{E}^k = \mathbf{D}^{-1}$, $\omega = 1$, $\mathbf{K}^k = \mathbf{0}$ and $\lambda = 1$ in equation (4.82):

$$\mathbf{x}^{k+1} = \mathbf{D}^{-1}(\mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x}^k). \tag{4.83}$$

In order to extend the general iterative method to LCPs we need a clamping operation that for a vector $\mathbf{x} \in \mathbb{R}^n$ and lower limits $\mathbf{lo} \leq \mathbf{0}$ and upper limits $\mathbf{hi} \geq \mathbf{0}$ with $\mathbf{lo}, \mathbf{hi} \in \mathbb{R}^n$ we can write $(\mathbf{x})^+$ to denote:

$$x_j^+ = \max(\min(x_j, hi_j), lo_j) \; \forall j \in \{0...n-1\}. \tag{4.84}$$

With the concept of clamping we can write a general iterative LCP solver as:

$$\mathbf{x}^{k+1} = \lambda(\mathbf{x}^k - \omega \mathbf{E}^k(\mathbf{A}\mathbf{x}^k - \mathbf{b} + \mathbf{K}^k(\mathbf{x}^{k+1} - \mathbf{x}^k)))^+ + (1-\lambda)\mathbf{x}^k. \tag{4.85}$$

Murty [59] has shown that if $\mathbf{A}$ is symmetric positive-definite the general scheme converges to a solution of the corresponding LCP. For detailed derivation and proofs of the general iterative LCP scheme we refer to [59]. Just as with the general form of solvers for a system of linear equations we can derive the Jacobi type solvers from it. For the Gauss-Seidel iterative LCP solver we choose $\lambda, \omega = 1$, $\mathbf{K}^k = \mathbf{L}$ and $\mathbf{E}^k = \mathbf{D}^{-1}$:

$$\mathbf{x}^{k+1} = (\mathbf{D}^{-1}(\mathbf{b} - (\mathbf{L}\mathbf{x}^{k+1} + \mathbf{U}\mathbf{x}^k)))^+. \tag{4.86}$$

The LCP solver variants of the iterative linear equation solvers are called *Projected Jacobi* (PJ), *Projected Gauss-Seidel* (PGS) and *Projected SOR* (PSOR) because the clamping operation can also be interpreted as a projection operation.

4.4.4 DEM-Based Contact Force Calculation

The discrete element method (DEM) is used in several particle simulation applications i.e. granular material simulation. The main idea of the method is to model soft sphere

collisions by applying a spring model for particle compression and decompression on contact. The method for granular materials is described in the works of Mishra [57], a general overview of the concepts of the DEM is given by [50, 66]. At its core the DEM is concerned only with spherical particles and not with arbitrarily shaped rigid bodies, this is why we will take a look at the basic model first and then show the extension of the DEM approach to handle rigid body simulations. The forces collision between two spherical particles is computed be considering the simple contact model shown in figure 4.9. A collision normal is constructed based on the difference vector between

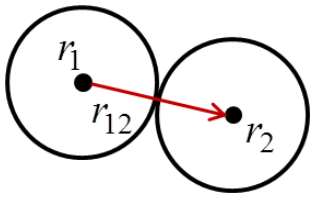


Figure 4.9: Simple contact information in the DEM.

the two sphere centers. In the DEM a small particle overlap is used to model the soft sphere properties of compression and decompression. The model that we use here was proposed by Harada [39], who had the goal in mind to create a rigid body simulation method that is well-suited to implementation on GPU hardware. The DEM models a soft sphere collision by calculating several force components. A repulsive force in normal direction is calculated by:

$$F_{i,s} = -k(d - |\mathbf{r}_{ij}|)\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|}, \tag{4.87}$$

here $F_{i,s}$ is the force at the contact between the i-th and j-th particle in normal direction which is described by a linear spring model. The parameters of the linear spring model are the spring coefficient k, the dampening coefficient η, the particle radius d, the relative position $\mathbf{r}_{ij}$ (and $|\mathbf{r}_{ij}|$ is the contact normal in the DEM context) and the relative velocity $\mathbf{v}_{ij}$. In the DEM tangential forces are modeled by:

$$F_{i,t} = k_t \cdot \mathbf{u}_{ij,t} \tag{4.88}$$

where k_t is the friction coefficient and $\mathbf{u}_{ij,t}$ is the relative tangential velocity:

$$\mathbf{u}_{ij,t} = \mathbf{u}_{ij} - (\mathbf{u}_{ij} \cdot \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|})\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|}. \tag{4.89}$$

Additionally, the DEM introduces a dampening force:

$$F_{i,d} = \eta \mathbf{u}_{ij}. \tag{4.90}$$

In the DEM the contact forces for the i-th particle are gathered from all surrounding particles that fulfill the contact condition which is easily evaluated for spherical particles. In the context of our simulation framework this would be the particles in the cells

of our broad phase grid that are neighbors of the cell that the i-th particle is in. The total forces acting on a particle are then evaluated by:

$$F_{i,c} = \sum_{collisions(i)} (F_{i,s} + F_{i,d} + F_{i,t}) \tag{4.91}$$

$$T_{i,c} = \sum_{collisions(i)} (\mathbf{r}_i \times (F_{i,s} + F_{i,d} + F_{i,t})). \tag{4.92}$$

The contact force calculation with the DEM is described in algorithm 7. The computation of contact forces with the DEM is done only in a pairwise manner and thus requires a smaller time step for the particle or rigid body solver then the LCP or sequential impulse based methods in order to accomplish proper propagation of shocks. The DEM can be extended to rigid bodies by representing rigid bodies by inner spheres (see figure 4.10). The details of this extension are found in the works of Harada [39].

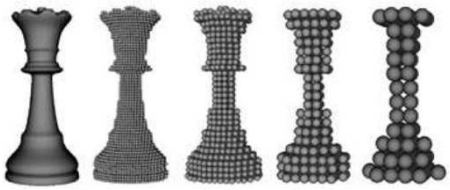


Figure 4.10: Approximation of rigid bodies by spheres

Algorithm 7: Contact force computation with the DEM

```
Data: ContactGraph G, Particles P
Result: Contact force for all particles
begin
    foreach particleNode p in G do
        foreach edge e of p in G do
            p.force += CalculateForce(e)
```

4.4.5 Sequential Impulses Model

The idea of sequential impulses is described in the works of Guendelman [37] and Catto [14]. The model is related to the LCP formulation in the sense that it also handles multiple contact points in one time step, but it does so in a sequential manner. The idea here is to do multiple sequential iterations of the pairwise impulse calculation that we showed before. This allows forces from dependent contact points to propagate through linked contact points (see figure 4.11). The model applies a normal force by computing

a normal impulse $\mathbf{f}$ for each contact point where the impulse magnitude f is calculated as in equation (4.42). The normal impulse is then:

$$\mathbf{f} = \frac{f\mathbf{n}}{m}, \tag{4.93}$$

where $\mathbf{n}$ is the collision normal of the contact point and m the mass of the rigid body that the impulse is applied to. The translational and angular velocity are then updated as in equation (4.41). The normal force will then keep the rigid bodies separated and prevent penetrations.

Friction in the Sequential Impulses Model

Frictional forces do not act in normal direction, but in tangential direction. In the sequential impulses model we would need a way to express a frictional force by an impulse. How to compute an impulse that represents a frictional force can be found in the works of Hahn [38]. This is done by a modified update of the pre-impulse relative velocity to the post-impulse relative velocity for a pair (A, B) of rigid bodies:

$$\mathbf{v}_{AB,+} = \mathbf{v}_{AB,-} + \mathbf{K}\mathbf{f} \tag{4.94}$$

$$\mathbf{f} = \mathbf{K}^{-1}(\mathbf{v}_{AB,+} - \mathbf{v}_{AB,-}). \tag{4.95}$$

Thus, the matrix $\mathbf{K}$ needs to be computed to arrive at the impulse:

$$\begin{aligned}
\mathbf{v}_{AB,+} - \mathbf{v}_{AB,-} &= (\mathbf{v}_{B,+} + \boldsymbol{\omega}_{B,+} \times \mathbf{r}_B - \mathbf{v}_{A,+} + \boldsymbol{\omega}_{A,+} \times \mathbf{r}_A) \\
&\quad - (\mathbf{v}_{B,-} + \boldsymbol{\omega}_{B,-} \times \mathbf{r}_B - \mathbf{v}_{A,-} + \boldsymbol{\omega}_{A,-} \times \mathbf{r}_A) \\
&= (\mathbf{v}_{B,+} - \mathbf{v}_{B,-} + (\boldsymbol{\omega}_{B,+} - \boldsymbol{\omega}_{B,-}) \times \mathbf{r}_B) \\
&\quad - (\mathbf{v}_{A,+} - \mathbf{v}_{A,-} + (\boldsymbol{\omega}_{A,+} - \boldsymbol{\omega}_{A,-}) \times \mathbf{r}_A) \\
&\overset{4.42}{=} (\frac{\mathbf{f}}{m_B} + (\mathbf{I}_B^{-1}(\mathbf{r}_B \times \mathbf{f})) \times \mathbf{r}_B) - (\frac{-\mathbf{f}}{m_A} + (\mathbf{I}_A^{-1}(\mathbf{r}_A \times -\mathbf{f})) \times \mathbf{r}_A) \\
&= \frac{1}{m_B}\mathbf{f} - (\mathbf{r}_B^{\times}\mathbf{I}_B^{-1}\mathbf{r}_B^{\times})\mathbf{f} + \frac{1}{m_A}\mathbf{f} - (\mathbf{r}_A^{\times}\mathbf{I}_A^{-1}\mathbf{r}_A^{\times})\mathbf{f} \\
&= (\underbrace{\frac{1}{m_B}\mathbf{1}_{3\times3} - \mathbf{r}_B^{\times}\mathbf{I}_B^{-1}\mathbf{r}_B^{\times}}_{\mathbf{K}_B} + \underbrace{\frac{1}{m_A}\mathbf{1}_{3\times3} - \mathbf{r}_A^{\times}\mathbf{I}_A^{-1}\mathbf{r}_A^{\times}}_{\mathbf{K}_A})\mathbf{f} \\
&= (\mathbf{K}_B + \mathbf{K}_A)\mathbf{f} = \mathbf{K}\mathbf{f}.
\end{aligned} \tag{4.96}$$

Then Hahn assumes that friction is static without tangential movement:

$$\mathbf{v}_{AB,+} = -\epsilon(\mathbf{v}_{AB,-}\mathbf{n})\mathbf{n}. \tag{4.97}$$

Thus the impulse acts not only in normal direction, but also in tangential direction and we can write:

$$\mathbf{f} = \mathbf{K}^{-1}(\mathbf{v}_{AB,+} - \mathbf{v}_{AB,-}) = \mathbf{K}^{-1}(-\epsilon\mathbf{v}_{AB,-}\mathbf{n} - \mathbf{v}_{AB,-}). \tag{4.98}$$

We can then extract the normal and tangential components of the impulse by:

$$\mathbf{f} = \mathbf{f}_n + \mathbf{f}_t \tag{4.99}$$
$$\mathbf{f}_n = (\mathbf{fn})\mathbf{n} \tag{4.100}$$
$$\mathbf{f}_t = \mathbf{f} - \mathbf{n}. \tag{4.101}$$

The assumption that friction is static is only valid if:

$$\mathbf{f}_t \leq \mu \|\mathbf{f}_n\|. \tag{4.102}$$

If equation (4.102) is not fulfilled then the impulse $\mathbf{f}$ is not in the friction cone and thus not valid. Hahn uses dynamic friction in this case by clamping the friction component to the maximally allowed dynamic friction:

$$\mathbf{f} = \mathbf{f}_n + \mu \|\mathbf{f}_n\| \frac{\mathbf{f}_t}{\|\mathbf{f}_t\|}. \tag{4.103}$$

An alternative to Hahn's approach has been presented by Guendelman [37] who uses a different method to compute friction in case that the static friction condition is not met:

$$\mathbf{t} = \frac{\mathbf{v}_{AB,-} - \mathbf{v}_{AB,n}\mathbf{n}}{\|\mathbf{v}_{AB,-} - \mathbf{v}_{AB,n}\mathbf{n}\|}. \tag{4.104}$$

An impulse can then be computed by:

$$\mathbf{f}_t = f\mathbf{n} - \mu f\mathbf{t}, \tag{4.105}$$

where the task is now to find the impulse magnitude f which can be achieved by:

$$\mathbf{v}_{AB,-} = \mathbf{v}_{AB,n,-} + \mathbf{n}^T\mathbf{K}\mathbf{f}. \tag{4.106}$$

By substituting $\mathbf{f}$ by equation (4.105) and using $\mathbf{v}_{AB,+} = -\epsilon\mathbf{v}_{AB,n,-}\mathbf{n}$ we get:

$$f = \frac{-(\epsilon+1)\mathbf{v}_{AB,-}\mathbf{n}}{\mathbf{n}^T\mathbf{K}(\mathbf{n} - \mu\mathbf{t})} \tag{4.107}$$

Iteration

As we said in the sequential impulse method the computation of impulses is iterated multiple times in every time step to mimic the the behavior of simultaneous methods. The iteration process can be repeated for a fixed number of iterations or we could check an iteration criterion like the norm of the vector of all relative normal velocities. If this value is close enough to zero it is safe to step the simulation forward in time. The procedure how to compute contact forces with sequential impulses in described in algorithm 8.
Theoretically, the sequential impulses model should yield similar behavior as the LCP formulation that we have mentioned before. The sequential impulses model is based

Algorithm 8: Contact force computation with sequential impulses

```
Data: ContactGraph G
Result: Contact force for all rigid bodies
begin
    while EvaluateConvergenceCriterion() not TRUE do
        foreach rigidBodyEdge e in G do
            foreach ContactPoint cp in e do
                cp.force += CalculateForce(e, cp)
```

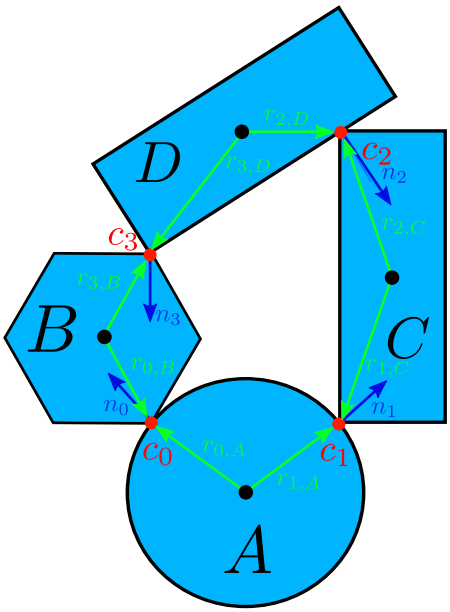


Figure 4.11: Dependent contact points in the sequential impulses method

on the same normal, tangential and frictional force computations. The difference between the approaches lies in the solving procedure, whereas in the constrained-based formulation a linear complementarity problem is solved with an iterative solver that solves a system of coupled equations, in the sequential impulses version we evaluate the forces in a pairwise fashion and immediately apply the forces at the contact point to get the updated velocities. These pairwise updates are iterated in a loop until our convergence criterion is met. This procedure has an advantage over the LCP formulation which is that the pairwise treatment of the contact points is a breakdown into local subproblems. In the LCP formulation there is one global matrix and it is not immediately clear how this matrix can be logically broken down into smaller matrices. In the context of parallel computing an approach that can be easily broken down into subproblems is advantageous as there subproblems are the basis for the distribution of the computation to different compute nodes.

Chapter 5

FEATFLOW Solver Overview
Designing an Efficient Liquid-Solid Interface

5.1 Introduction

For the numerical solution of the fluid flow in our particulate flow simulation we use the FEATFLOW solver. The FEATFLOW solver package is a parallel solver for the incompressible Navier-Stokes equation and is able to handle Newtonian, non-Newtonian and viscoelastic fluids. For many realistic applications it is necessary not only to simulate the fluid flow, but also some kind of immersed geometry. In order to handle solid geometries in a fluid some kind of *interface capturing* or *interface tracking* method is needed. In the FEATFLOW solver this interface method is the fictitious boundary method (FBM) [88]. When we explore the history of particulate flow simulations we see that several numerical simulation techniques for particulate flow have been developed in the past. In these methods, the fluid flow is governed by the continuity and momentum equations, whereas for the particles the governing equations are the Newton-Euler equations of motion. Another general aspect found in many particulate flow solving schemes is the computation of the hydrodynamic force between the particle and the fluid. For an accurate computation of these forces the fluid flow needs to be finely resolved by the mesh around the immersed rigid bodies. The historic origins of finite element based particulate flow simulations in Newtonian and viscoelastic fluids can be found in the works of Hu, Joseph et al. [40, 41], Galdi [28] or Maury [51]. Their approach can be characterized by the use of unstructured grids and an *Arbitrary Lagrangian-Eulerian* (ALE) technique. Then both the fluid and solid equations of motion are written as a single coupled variational formulation. When as a consequence of the ALE the mesh becomes too distorted at a certain time step a new mesh is generated

and the flow field is projected onto the new mesh. In such schemes, the positions of the particles and grid nodes are updated explicitly, while the velocities of the fluid and the solid particles are determined implicitly.
Another approach is based on so called *fictitious domains*, an idea that was originally introduced by Glowinski, Joseph, Patankar et al. [32, 63, 31, 33]. In this Eulerian method the presence of an additional domain inside the computational domain is simulated, the fictitious domain. We thus introduce a boundary between these domains, this boundary is mathematically realized by adding appropriate terms to the model. In this approach to particulate flow simulation, the particle domain is treated as a fluid with additional constraints to impose rigid body motion inside of the fictitious domain. Several different implementations of this general principle exist, the variants differ mainly in the mathematical realization of the boundary, the way the rigid motion is imposed on the particle domain or in the numerical solution process of the problem. A common ground of all the fictitious domain type methods is that they allow the use of a fixed grid which does not require remeshing. The idea goes back to to Glowinski, Joseph, Patankar et al. [32, 63, 31, 33], who proposed a approach based on a distributed Lagrange multiplier (DLM)/fictitious domain method for the direct numerical simulation of large number of rigid particles in fluids. In the DLM method, the entire fluid-particle domain is assumed to be a fluid and then the particle domain is constrained to rigid body motion. The fluid-particle coupling is treated implicitly using a combined weak formulation in which the mutual forces cancel [42]. The following work by Patankar and Sharma [64, 78] represents an improvement in terms of solution time, which was achieved by an efficient projection scheme. The fictitious domain method is also used in the 3D particulate flow simulations of Boenisch [10], Cottet/Coquerelle [17] and Blasco [9].

The FEATFLOW group developed and refined the multigrid fictitious boundary method (FBM) for the direct simulation of particulate flows [89, 88, 90, 58]. In the multigrid FBM the motion of solid particles is modeled by the Newton-Euler equations. Based on the boundary conditions applied at the interface between the particles and the fluid which can be seen as an additional constraint to the governing Navier-Stokes equations, the fluid domain can be extended into the whole domain which covers both fluid and particle domains. The FBM can be regarded as a fictitious domain method and it shares the feature that a fixed grid can be used, eliminating the need for remeshing. An underlying problem with fictitious domain methods that use a fixed mesh is that the boundary approximation is of low accuracy only. Especially in three space dimensions, the ability of the fictitious domain methods to deal with the interaction between fluid and rigid particles accurately is limited when the only way to achieve better geometry resolution is by regular refinement. A possible approach to improve the geometry resolution is to keep the structure of the mesh and to perform a local alignment of the mesh vertices with the physical boundary of the solid particles by a mesh adaptation method, such that the boundary approximation error can be significantly decreased.

Apart from finite element based methods also flow solvers based on the Lattice-Boltzmann

method [47] or Finite-Volume method can be used as fluid solvers in particulate flow scenarios [72].
In the context of particulate flow simulation one of the most important aspects of the flow solver is the accurate computation of the hydrodynamic forces. In particular these forces are the drag and the lift forces, which act on the surface of the rigid body. This gives rise to some problems, that have to be overcome in order to effectively use the FBM. These problems are that the surface of the rigid body is implicitly represented by a fixed mesh, so the quality of the surface approximation and consequently the accuracy of the drag/lift calculations depend on the refinement level of the grid. Furthermore, the surface is represented implicitly, so the surface integral formulation of the drag and lift forces cannot be used directly in the FBM framework. In the 2D case our remedy for the latter problem was to integrate over the area occupied by the rigid body rather than integrating over the boundary of the rigid body. In the 3D case this generalizes to a volume integral formulation. Furthermore, the accuracy of the geometry shape representation by the mesh can be improved by mesh adaptation methods, which are able to significantly improve results in the 2D case [90]. In the case of moving rigid bodies a moving mesh formulation of the Navier-Stokes equations is needed.

Another important aspect for particulate flow simulation is collision treatment which traditionally in the FEATFLOW solver was handled by a repulsive force model [89]. In this work all the collision handling and detection in the FEATFLOW solver will be replaced by the methods in chapter 3. Other collision models that are used in different particulate flow solver frameworks include the work of Maury [52], Ardekani/Rangel [2] or Lefebvre [48].

5.2 Governing Equations for Fluid Flow

In our numerical particulate flow simulations, we assume that the fluids are immiscible and Newtonian. Furthermore, we assume that the particles are not-deformable and thus rigid. We can then consider the flow of N rigid particles with masses M_i $(i = 1, \ldots, N)$ in a fluid with density ρ_f and viscosity ν. By $\Omega_f(t)$ we identify the domain occupied by the fluid at time t, and by $\Omega_i(t)$ the volume of the domain by the i-th particle. We can then describe the motion of an incompressible fluid in $\Omega_f(t)$ by the Navier-Stokes equations,

$$\rho_f \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) - \nabla \cdot \sigma = 0 \,, \qquad \nabla \cdot \mathbf{u} = 0 \qquad \forall\, t \in (0, T), \tag{5.1}$$

where σ is the total stress tensor in the fluid phase defined as

$$\sigma = -p\, \mathbf{I} + \mu_f \left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right]. \tag{5.2}$$

Where $\mathbf{I}$ is the identity tensor, μ_f the dynamic viscosity, p the pressure and $\mathbf{u}$ is the fluid velocity. We denote by $\Omega_T = \Omega_f(t) \cup \{\Omega_i(t)\}_{i=1}^N$ the entire computational do-

main which shall be independent of t. Dirichlet- and Neumann-type boundary conditions can be imposed on the outer boundary $\Gamma = \partial\Omega_f(t)$. Since $\Omega_f = \Omega_f(t)$ and $\Omega_i = \Omega_i(t)$ always depend on t, we will omit it in the following equations. The equations that govern the motion of each particle are the Newton-Euler equations, i.e., the translational velocities $\mathbf{U}_i$ and angular velocities $\boldsymbol{\omega_i}$ of the i-th particle are given by:

$$M_i \frac{d\,\mathbf{U}_i}{d\,t} = (\Delta M_i)\,\mathbf{g} + \mathbf{F}_i + \mathbf{F}'_i\,, \qquad \mathbf{I}_i \frac{d\,\boldsymbol{\omega_i}}{d\,t} + \boldsymbol{\omega_i} \times (\mathbf{I}_i\,\boldsymbol{\omega_i}) = \mathbf{T}_i\,, \tag{5.3}$$

where M_i is the mass of the i-th particle; $\mathbf{I}_i$ is the moment of inertia tensor of the i-th particle about its center of mass; ΔM_i is the mass difference between the mass M_i and the mass of the fluid occupying the same volume; $\mathbf{g}$ is the gravity vector; $\mathbf{F}'_i$ are collision forces acting on the i-th particle due to other particles which come close to each other. $\mathbf{F}_i$ and $\mathbf{T}_i$ are the hydrodynamic forces and the torque about the center of mass acting on the i-th particle which are calculated by

$$\mathbf{F}_i = (-1)\int_{\partial\Omega_i} \sigma\cdot\mathbf{n}\,d\,\Gamma_i\,, \qquad \mathbf{T}_i = (-1)\int_{\partial\Omega_i} (\mathbf{X}-\mathbf{X}_i)\times(\sigma\cdot\mathbf{n})\;d\,\Gamma_i, \tag{5.4}$$

where σ is the total stress tensor in the fluid phase defined by equation (5.2), $\mathbf{X}_i$ is the position of the mass center of the i-th particle, $\partial\Omega_i$ is the boundary of the i-th particle, $\mathbf{n}$ is the outward pointing normal vector of boundary $\partial\Omega_i$. The position $\mathbf{X}_i$ of the i-th particle is obtained by integration of the kinematic equation

$$\frac{d\,\mathbf{X}_i}{d\,t} = \mathbf{U}_i. \tag{5.5}$$

The angular velocity $\boldsymbol{\omega_i}$ and the angle $\boldsymbol{\theta_i}$ are calculated from the angular acceleration a_i and torque $\mathbf{T}_i$ of the i-th particle using the following equations:

$$\mathbf{T}_i = \mathbf{I}_i a_i \tag{5.6}$$

$$\frac{d\boldsymbol{\omega_i}}{dt} = a_i \tag{5.7}$$

$$\frac{d\boldsymbol{\theta_i}}{dt} = \boldsymbol{\omega_i} \tag{5.8}$$

We plug equation (5.7) into equation (5.6), then multiply by the moment of inertia tensor $\mathbf{I}_i^{-1}$ and integrate the torque to get the angular velocity ω_i.The angle θ_i can hence be calculated by integrating the angular velocity. No-slip boundary conditions are applied at the interface $\partial\Omega_i$ between the i-th particle and the fluid, i.e., for any $\mathbf{X} \in \bar{\Omega}_i$, the velocity $\mathbf{u}(\mathbf{X})$ is defined by

$$\mathbf{u}(\mathbf{X}) = \mathbf{U}_i + \boldsymbol{\omega_i} \times (\mathbf{X} - \mathbf{X}_i)\,. \tag{5.9}$$

5.3 Numerical Method

5.3.1 Multigrid FEM-FBM

The multigrid FEM-FBM [88, 91, 87] is based on a multigrid FEM background grid which covers the whole computational domain Ω_T and can be chosen independently from the particles of arbitrary shape, size and number. It starts with a coarse mesh which may already contain many of the geometrical details of Ω_i $(i = 1, \ldots, N)$, and it employs a fictitious boundary indicator (see [87]) which sufficiently describes all fine-scale structures of the particles with regard to the fluid-particle matching conditions of equation (5.9). Then, all fine-scale features of the particles are treated as interior objects such that the corresponding components in all matrices and vectors are unknown degrees of freedom which are implicitly incorporated into all iterative solution steps (see [91]). Hence, by making use of equation (5.9), we can perform calculations for the fluid in the whole domain Ω_T. The considerable advantage of the multigrid FBM is that the total mixture domain Ω_T does not have to change in time, and can be meshed only once. The domain of definition of the fluid velocity $\mathbf{u}$ is extended according to equation (5.9), which can be seen as an additional constraint to the Navier-Stokes equations (5.1), i.e.,

$$\begin{cases} \nabla \cdot \mathbf{u} = 0 & (a) \quad \text{for } \mathbf{X} \in \Omega_T, \\ \rho_f \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) - \nabla \cdot \sigma = 0 & (b) \quad \text{for } \mathbf{X} \in \Omega_f, \\ \mathbf{u}(\mathbf{X}) = \mathbf{U}_i + \omega_i \times (\mathbf{X} - \mathbf{X}_i) & (c) \quad \text{for } \mathbf{X} \in \bar{\Omega}_i,\ i = 1, \ldots, N. \end{cases} \tag{5.10}$$

For the study of interactions between the fluid and the particles, the calculation of the hydrodynamic forces acting on the moving particles is very important. From equation (5.4), we can see that surface integrals over the surfaces of the particles appear in the calculation of the forces $\mathbf{F}_i$ and T_i. However, in the presented multigrid FBM method, the shapes of the surfaces of the moving particles are implicitly imposed in the fluid field. If we reconstruct the shapes of the surface of the particles, it is not only a time consuming work, but also the accuracy is only of first order due to a piecewise constant interpolation from our indicator function. In order to resolve the shape, we perform the hydrodynamic force calculations using a volume based integral formulation. To replace the surface integral in equation (5.4) we introduce a function α_i,

$$\alpha_i(\mathbf{X}) = \begin{cases} 1 & \text{for} \quad \mathbf{X} \in \Omega_i, \\ 0 & \text{for} \quad \mathbf{X} \in \Omega_T \setminus \Omega_i \end{cases} \tag{5.11}$$

where $\mathbf{X}$ denotes the coordinates. An interesting property of this function is that the gradient of α_i is zero everywhere except at the surface of the i-th particle, and approximates the normal vector $\mathbf{n}_i$ of surface of the i-th particle in a weak sense. This allows us to write:

$$\mathbf{F}_i = -\int_{\Omega_T} \sigma \cdot \nabla \alpha_i \, d\Omega, \qquad T_i = -\int_{\Omega_T} (\mathbf{X} - \mathbf{X}_i) \times (\sigma \cdot \nabla \alpha_i) \, d\Omega \,. \tag{5.12}$$

In the finite element framework this can then be evaluated as

$$\mathbf{F}_i = -\sum_{T\in T_{h,i}} \int_{\Omega_T} \sigma_h \cdot \nabla \alpha_{h,i}\, d\,\Omega, \qquad T_i = -\sum_{T\in T_{h,i}} \int_{\Omega_T} (\mathbf{X}-\mathbf{X}_i)\times(\sigma_h\cdot\nabla\alpha_{h,i})\; d\,\Omega \tag{5.13}$$

where $\alpha_{h,i}(\mathbf{X})$ is the finite element interpolant of $\alpha(\mathbf{X})$ and $T_{h,i}$ the elements that are intersected by the i-th particle.

5.3.2 Time Discretization by Fractional-Step-θ Scheme

The fractional-step-θ scheme is a strongly A-stable time stepping approach which has the full smoothing property that is important in the case of rough initial or boundary data [85]. We first semi-discretize the equations (5.10) (a) and (5.10) (b) in time by the fractional-step-θ scheme. Given $\mathbf{u}^n$ and the time step $K = t_{n+1} - t_n$, then solve for $\mathbf{u} = \mathbf{u}^{n+1}$ and $p = p^{n+1}$. In the fractional-step-θ-scheme, one macro time step $t_n \to t_{n+1} = t_n + K$ is split into three consecutive substeps with $\tilde{\theta} := \alpha\theta K = \beta\theta' K$,

$$\begin{aligned}
[I+\tilde{\theta}N(\mathbf{u}^{n+\theta})]\mathbf{u}^{n+\theta} + \theta K\nabla p^{n+\theta} &= [I-\beta\theta K N(\mathbf{u}^n)]\mathbf{u}^n \\
\nabla\cdot\mathbf{u}^{n+\theta} &= 0\,, \\
[I+\tilde{\theta}N(\mathbf{u}^{n+1-\theta})]\mathbf{u}^{n+1-\theta} + \theta' K\nabla p^{n+1-\theta} &= [I-\alpha\theta' K N(\mathbf{u}^{n+\theta})]\mathbf{u}^{n+\theta} \\
\nabla\cdot\mathbf{u}^{n+1-\theta} &= 0\,, \\
[I+\tilde{\theta}N(\mathbf{u}^{n+1})]\mathbf{u}^{n+1} + \theta K\nabla p^{n+1} &= [I-\beta\theta K N(\mathbf{u}^{n+1-\theta})]\mathbf{u}^{n+1-\theta} \\
\nabla\cdot\mathbf{u}^{n+1} &= 0\,,
\end{aligned} \tag{5.14}$$

where $\theta = 1 - \frac{\sqrt{2}}{2}$, $\theta' = 1-2\theta$, and $\alpha = \frac{1-2\theta}{1-\theta}$, $\beta = 1-\alpha$, $N(\mathbf{v})\mathbf{u}$ is a compact form for the diffusive and convective part,

$$N(\mathbf{v})\mathbf{u} := -\nu\;\nabla\cdot\left[\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right] + \mathbf{v}\cdot\nabla\mathbf{u}\,. \tag{5.15}$$

Therefore, for equation (5.14), we have to solve in each time step nonlinear problems of the type:

$$[I+\theta_1 K N(\mathbf{u})]\mathbf{u} + \theta_2 K\nabla p = \mathbf{f}\,, \quad \mathbf{f} := [I-\theta_3 K N(\mathbf{u}^n)]\mathbf{u}^n\,, \;\; \nabla\cdot\mathbf{u} = 0\,. \tag{5.16}$$

For equation (5.10) (c), we take an explicit expression:

$$\mathbf{u}^{n+1} = \mathbf{U}_i^n + \omega_i^n \times (\mathbf{X}^n - \mathbf{X}_i^n)\,. \tag{5.17}$$

5.3.3 Space Discretization by Finite Element Method

For the spatial discretization we choose a finite element approach that is based on a suitable variational formulation [86]. We introduce a finite element mesh T_h consisting of hexahedrons to cover the whole computational domain Ω , where h characterizes the maximum edge length of the elements of T_h. To obtain the fine mesh T_h from a coarse

mesh T_{2h}, we simply apply a regular refinement to the hexahedral cells that splits each hexahedron into 8 new hexahedrons. We define polynomial trial functions for velocity and pressure, for the corresponding spaces H_h and L_h we should achieve numerically stable approximations for $h \to 0$, meaning they should satisfy the $inf\text{-}sup$ (LBB) condition [30]

$$\min_{q_h \in L_h} \max_{\mathbf{v}_h \in H_h} \frac{(q_h, \nabla \cdot \mathbf{v}_h)}{\|q_h\|_0 \|\nabla \mathbf{v}_h\|_0} \geq \gamma > 0 \tag{5.18}$$

with a mesh-independent constant γ. As finite element we choose the $Q2/P1$ element pair for space discretization and as FEM stabilization techniques we use edge-, resp., face-oriented stablization. For further details on the discretization and solution approach we refer to [86] and to [62, 20] for details of the stabilization for the $Q2/P1$ element pair.

5.4 Liquid-Solid Interface

5.4.1 Introduction

As explained earlier the nodes of the computational mesh need to be marked as solid or liquid nodes as part of creating the indicator (see equation (5.11)) function before the physical equations can be solved. A node is classified as a solid node in case it is located inside the solid geometry or on its surface. The problem (see figure 5.1) is relatively simple to solve in case we are dealing with a regular or structured mesh and simple solid geometries like spheres or ellipsoids. A structured quadrilateral or hexahedral mesh is defined as a subdivision of 2- or 3-dimensional Euclidean space by congruent quadrilaterals or hexahedrons. In the 2D case every cell of the mesh can be addressed by a pair of indices (i, j). The coordinates of the nodes of the mesh are $(i \cdot \Delta x, j \cdot \Delta y)$, as a datastructure to store such structured 2D mesh usually 2D arrays are chosen with a direct mapping between array indices and cell indices. Let us assume we want to solve the node classification problem for a mesh with n nodes and m solid bodies. The most simple and most naive way of solving the node classification problem would loop over all nodes of the mesh and then compute for every solid if the node is inside the solid body. The complexity of this naive approach is $O(n \cdot m)$.
The naive approach can be improved by using the direct relation between the node indices and the coordinates of the node. If we assign a minimal bounding box to a solid it is sufficient to only consider the nodes that are inside of the bounding box as candidates for nodes that could be inside of the real solid geometry. The pseudo-code description for this simple point classification method is outlined in algorithm 9. This way the number of nodes that need to be checked for each solid object can be reduced to a constant number, the complexity of this the point classification method is then reduced to $O(m)$. In these considerations the algorithmic complexity of determining whether the coordinates of a node of the mesh are located inside the solid object are considered to be constant which is valid for most simple geometric shapes.

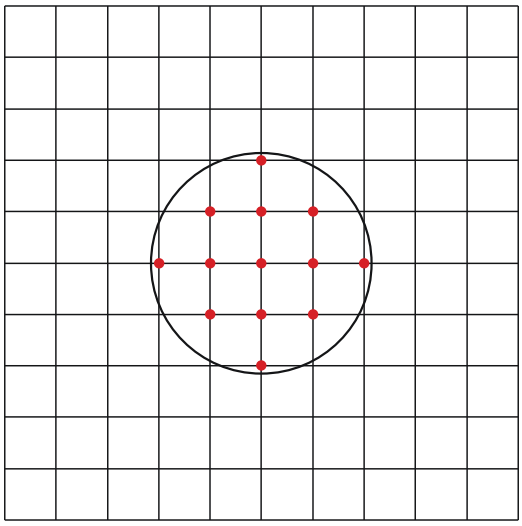


Figure 5.1: Illustration of the point classification problem, the nodes inside of the circle need to be marked as solid nodes

5.4.2 Fast Point Location in Unstructured Meshes

The point classification problem is more complex for the class of unstructured meshes. Unstructured meshes are characterized by an irregular connectivity, such that a node of the mesh can have any number of neighboring nodes. The complications arise from the fact that for an unstructured mesh there is no mapping from the geometric location of the nodes to indices in the mesh data structure. The data structure used in FEATFLOW to store an unstructured mesh is in essence a list-based data structure where for each element of the mesh a list of neighbors is stored. Additional complication arises from the domain decomposition approach that is used in FEATFLOW, so that in fact we are dealing with submeshes in each domain which as a union form the whole domain. This decomposition introduces additional irregularity in terms of the geometric shape of the subdomains and the number of neighboring elements in the boundary region of two subdomains. As we can see in figure 5.2 the geometrical shape of the subdomains is not limited to regular shapes like axis-aligned boxes or similar. The standard FEATFLOW mesh data structure does not allow us to efficiently traverse nodes as was the case in the example of the structured grid. Without extended information this would require checking all n nodes of our mesh to solve the point classification problem. In algorithm 9 we saw that using a bounding box representation of a solid object provided us with the possibility to check only a small number of nodes for each solid object. In order to use this approach for unstructured meshes we need to introduce an additional search grid that allows us to quickly determine the elements of the mesh that intersect with the bounding box representation of the solid object. A well suited choice of search

Algorithm 9: Simple Bounding Box Point Classification for Structured Meshes

Data: x0, x1, y0, y1, obj, mesh
Result: mesh.marker
begin

$x0 := \left\lfloor \frac{obj.boundingBox.x_{min}}{mesh.cellSize} \right\rfloor$
$x1 := \left\lceil \frac{obj.boundingBox.x_{max}}{mesh.cellSize} \right\rceil$
$y0 := \left\lfloor \frac{obj.boundingBox.y_{min}}{mesh.cellSize} \right\rfloor$
$y1 := \left\lceil \frac{obj.boundingBox.y_{max}}{mesh.cellSize} \right\rceil$
for $i = x0$ *to* $x1$ **do**
  for $j = y0$ *to* $y1$ **do**
    if $obj.$*PointInObject*$(mesh.nodes[i][j])$ **then**
      $mesh.marker[i][j] :=$ solid

grid would be the hash grid data structure which we have introduced in chapter 3. We use a hash grid for each subdomain of the decomposition and set its bounding box to the bounding box of the subdomain. As we have seen in figure 5.2 the shapes of the subdomain can be irregular and thus some parts of the subdomain bounding box are not filled with cells in the particular subdomain. An advantage of using the hash grid data structure is that no cells will be created in this empty region and thus no memory is wasted. We can then insert the vertices or cells (as needed) into the search grid and then perform the same procedure as in algorithm 9 in each of the subdomains in parallel.

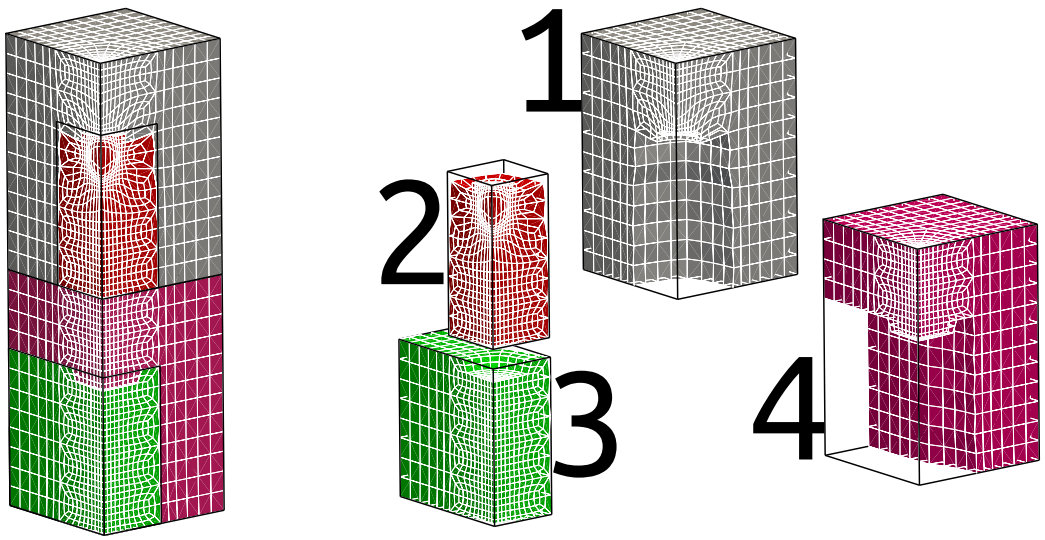


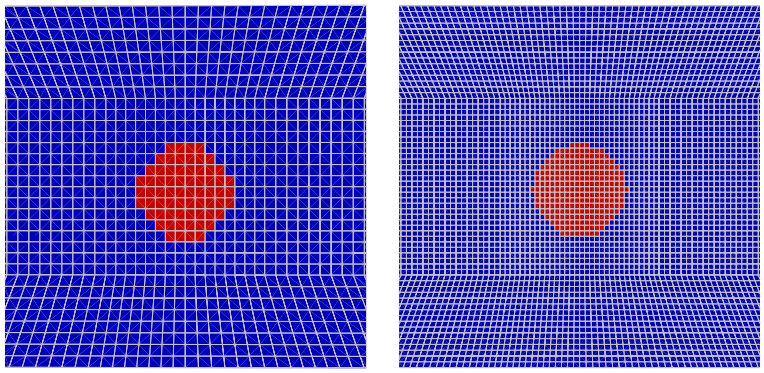
Figure 5.2: Domain Decomposition in FEATFLOW: Whole domain (left), domain decomposed in 4 subdomains (right)

5.4.3 GPU-based Point Location in Unstructured Meshes

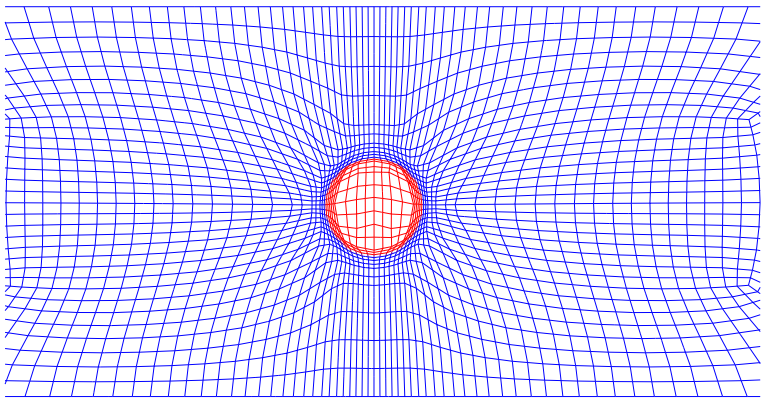
If GPU hardware is available then the *PointInObject* test can be performed efficiently on the GPU. The only change that needs to be done for a GPU version is to generate data structures that allow an efficient implementation of the *PointInObject* test on the GPU like the distance map data structure or the inner sphere data structure (see chapter 3). We can then depending on the application setup either manage the hash grid data structure on the GPU or we can transfer the points of the computational mesh that are located inside the bounding box of the current object to the GPU and then launch a CUDA thread for every vertex of the mesh to perform the *PointInObject test*.

5.5 Introduction to Mesh Deformation

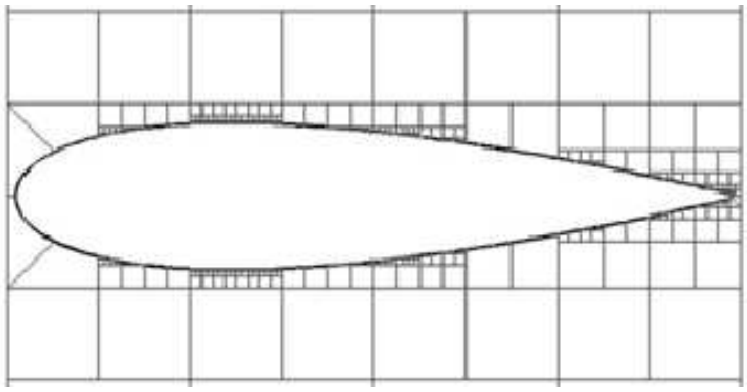
When performing calculations on computational meshes involving an immersed geometry the quality of the geometry resolution by the mesh is directly related to the accuracy of the corresponding calculation. In order to improve the quality of the geometry resolution one possible approach is to increase the resolution by regular refinement of the mesh. This however has as a consequence that in the case of a hexahedral mesh all element based calculations increase by a factor of 8 because a single cell is divided into 8 smaller cells. Another possibility is to locally refine the elements that represent the surface of the immersed geometry which only slightly increases the number of new elements because we do not refine globally. This approach which is called *h-adaptivity* creates hanging nodes between those elements which are refined and those which remain as they are. The introduction of hanging nodes requires special numerical treatment which is not implemented in all modules of the FEATFLOW solver so that this option may not always be available. Another mesh adaptation approach is to keep the mesh connectivity as it is, but to redistribute the existing nodes in a way that they resolve the immersed geometry better. This way of improving the mesh resolution of an object is called r-adaptation. In our work we will explore the class of r-adaptation techniques to improve the resolution of our geometries by the mesh. This type of mesh adaptations techniques often relies on the concept of a *monitor function*, a function that provides a mathematical description of the desired deformation. The design and calculation of monitor functions often involves computing the distance to the objects as this is a good criterion of adaptation because it is desirable to redistribute the nodes into those regions where the distance to the object is low. We can furthermore make a distinction between techniques that require additional partial differential equations to be solved and those that do not involve PDEs.



(a) Adaptation to a circle by regular refinement



(b) Adaptation to a circle by r-adaptivity



(c) Mesh adaptation to an airfoil by h-adaptivity

Figure 5.3: Different types of mesh adaptation

5.6 PDE-Based R-Adaptivity Mesh Deformation Algorithm

In this section we will briefly summarize the basic aspects of PDE-based mesh adaptation. Assume the domain Ω is covered by a regular grid $\mathcal{T}_0$. The aim of r-adaptivity

class grid deformation algorithms that follow the schemes of Moser [19], Liao [12] or Grajewski [35] is to to compute a bijective mapping $\Psi : \Omega \to \Omega$ that satisfies:

$$g_0(\mathbf{x})|J\Psi(\mathbf{x})| = f(\Psi(\mathbf{x})),\ \mathbf{x} \in \Omega \text{ and } \partial\Omega \to \partial\Omega \tag{5.19}$$

Here the function g_0 is the cell size distribution of the cells in the initial grid $\mathcal{T}_0$. Thus, the function g_0 is often referred to as the *area function*. The computed mapping Ψ performs a transformation $\Psi : \mathbf{x} \to \mathbf{x}_{new}$ of the grid points $x \in \mathcal{T}_0$ to new coordinates $\mathbf{x}_{new}$ and thus a transformation of the initial grid $\mathcal{T}_0$ to the target grid $\mathcal{T}_{goal}$. The function f is the so called *monitor function* which describes the desired cell size distribution on the target grid $\mathcal{T}_{goal}$. Examining equation (5.19) we notice that the term $|J\Psi(\mathbf{x})|$ represents the relative growth of the grid cells around the vertex $\mathbf{x}$ and the product $g_0(\mathbf{x})|J\Psi(\mathbf{x})|$ the absolute cell size distribution after deformation. The functions f and g_0 are both computed on the original grid $\mathcal{T}_0$. To compute the function g_0, we need cell size distribution values in the vertices $\mathbf{x}$ of the grid $\mathcal{T}_0$. To obtain these values we take the average cell size values of the elements that are incident to a vertex $\mathbf{x}$. Before we state the outline of the grid deformation algorithm, we still have to choose an appropriate monitor function f. According to Grajewski [35] a basic grid deformation algorithm can thus be stated as in algorithm 10. The gradient of v in eq. (5.22) we obtain by computing an approximate solution v_h using finite elements and applying a gradient recovery technique to it, which yields a recovered gradient $G_h(v_h)$. Grajewski's analysis [35] showed that the gradient recovery method has almost no influence on the result of the computed deformation. The initial value problem (IVP) in (5.22) can be solved by standard methods for example an explicit euler scheme, but the solution of the IVP involves a 'hidden' geometrical problem. Examining the IVP, we notice that in each time step the FEM functions $G_h(v_h)$, $\tilde{f}$ and $\tilde{g_0}$ have to be evaluated. These functions are defined on the initial grid $\mathcal{T}_0$ and in order to evaluate them, it is necessary to know in which element the grid point, in which we want to evaluate the functions is located. This element is known with certainty only in the first time step. After the first time step the grid point may have already moved to a different element. From this fact arises a *point location problem*. Specifically, this means that we have to search and find the element that a point is located in, in every time step but the first. This problem is easily solvable if $\mathcal{T}_0$ is a regular structured grid. In this case the element indices can be determined by analyzing the coordinates of the vertex. If $\mathcal{T}_0$ is an unstructured grid the task is more difficult and we will briefly revisit the most popular strategies for this task.

Brute Force: Starting with the element I_{old} that contained the point in the last time step we loop over all elements until we have found the element that contains the point. The complexity of this algorithm is $\mathcal{O}(NVT \cdot NEL)$ in the worst case. This algorithm cannot be used for any practically relevant problems as a standard search strategy, because of its unacceptable runtime. However, it may serve as a fallback strategy if everything else fails.

Raytracing Search: We start checking the old element I_{old} whether it still contains

Algorithm 10: Basic grid deformation algorithm (BDM)

Data: f,g_0
Result: Ψ
begin

1. Compute Scaling factors c_f and c_g for f and g_0 such that:

$$c_f \int_\Omega \frac{1}{f(\mathbf{x})} dx = c_g \int_\Omega \frac{1}{g_0(\mathbf{x})} = |\Omega| \tag{5.20}$$

2. Let $\tilde{f} = \frac{c_f}{f}$ and $\tilde{g_0} = \frac{c_g}{g_0}$ denote the reciprocal functions of f and g_0. Compute $v : \Omega \to \mathbb{R}$ by solving

$$-div(v(\mathbf{x})) = \tilde{f}(\mathbf{x}) - \tilde{g_0}(\mathbf{x}), \mathbf{x} \in \Omega \text{ and } v(\mathbf{x}) \cdot \mathbf{n} = 0 \text{ for } \mathbf{x} \in \partial\Omega, \tag{5.21}$$

where $\mathbf{n}$ is the outward pointing normal vector of the domain boundary $\partial\Omega$

3. Find the new position $\mathbf{x}_{new}$ for each grid point $\mathbf{x}$ by solving the following initial value problem

$$\frac{\varphi(\mathbf{x}, t)}{\partial t} = \eta(\varphi(\mathbf{x}, t), t), 0 \leq t \leq 1, \varphi(\mathbf{x}, 0) = \mathbf{x} \tag{5.22}$$

where $\eta(\mathbf{y}, s)$ is defined as

$$\eta(\mathbf{y}, s) := \frac{\nabla v(\mathbf{y})}{s\tilde{f}(\mathbf{y}) + (1-s)\tilde{g_0}(\mathbf{y})}, \mathbf{y} \in \Omega, s \in [0, 1] \tag{5.23}$$

4. Define $\Psi(\mathbf{x}) := \varphi(\mathbf{x}, 1) = \mathbf{x}_{new}$

the point. If this is not the case we connect the mid point of the old element with the new location of the grid point. We have thus created a search path along which we can track the path the grid point has moved and check the elements along this path. This is done by performing an intersection test between the faces of the current element and the search line. If the line intersects with a face we continue to track the line in the neighbor of the element in that face. This procedure is repeated until we have found the element that contains the grid point.

We will demonstrate the setup and application of the method to a simple test problem. The results of the numerical solution of the test problem with the BDM are illustrated in figure (5.4). The problem is formulated as follows:

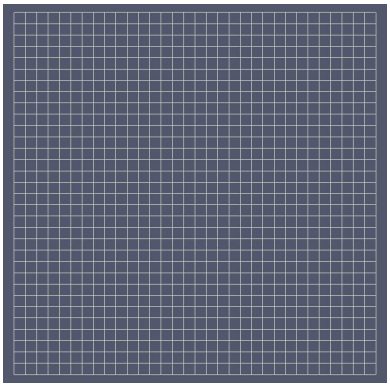
The unit square $\Omega = [0,1]^2$ is covered with a regular tensor product grid $\mathcal{T}_0$. Apply the BDM to $\mathcal{T}_0$ using the monitor function f:

$$f(\mathbf{x}) = \min\{1, \max\{\frac{\|d - 0.25\|}{0.25}, \varepsilon\}\} \tag{5.24}$$

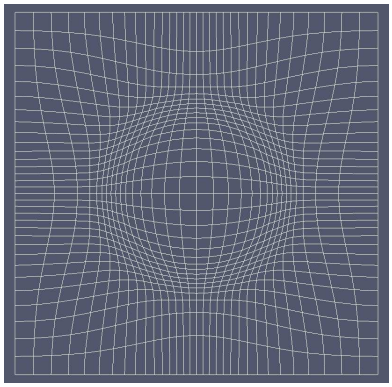
The parameter ε is the ratio of the smallest element to the largest element on the deformed grid and is set to 0.1 for the test problem. The choice of ε plays an important role for the adaptation process. Decreasing the so called *shape parameter* ε causes the algorithm to concentrate a higher number of grid points around the circle defined by the test monitor function. The BDM is likely to run into numerical problems because it is a one step method that can potentially prescribe a harsh deformation that may lead to solver problems for the Laplace problem or the IVP. Therefore in [35] a robust deformation method was proposed, that can break up a considerably harsh adaptation process in several less harsh *adaptation steps*. Algorithmically this is realized by introducing a *blended monitor function* f_s, where f_s is a linear combination of f and g_0:

$$f_s(\mathbf{x}) := sf(\mathbf{x}) + (1-s)g_0(\mathbf{x}),\ s \in [0,1] \tag{5.25}$$

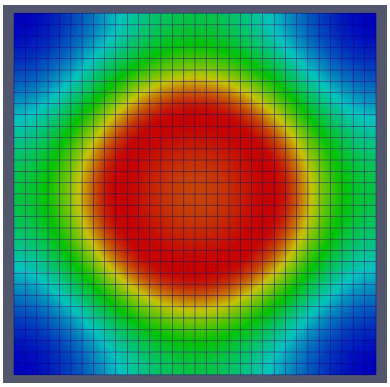
The *blending parameter* s is increased in every adaptation step, so that the blended monitor function f_s resembles the original f more and more with each adaptation step. Furthermore, the composition of the mappings computed in the adaptation steps, yields the desired mapping Ψ. For the computation of the individual adaptation steps we can resort to the BDM. The ideas and concepts discussed so far allow us to state the outline of the robust deformation method (RDM). A remedy for this it is possible to apply a grid smoothing technique like *laplacian smoothing* after the adaptation steps.



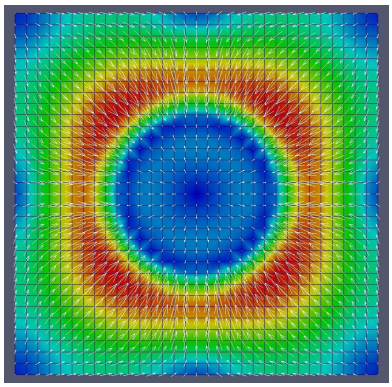
(a) The equidistant start grid $\mathcal{T}_0$



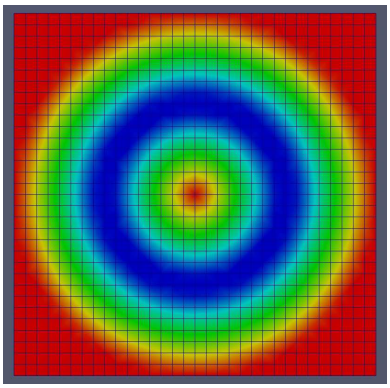
(b) Grid after application of the BDM



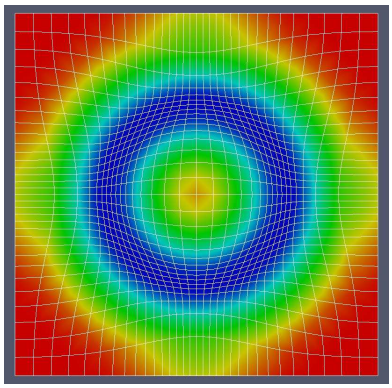
(c) Solution of equation (5.21)



(d) The recovered gradient of v_h



(e) Monitor function for the test problem



(f) Area distribution around the vertices of the deformed grid

Figure 5.4: Results of the BDM for the test problem ('grid adaption to a circle')

Algorithm 11: Robust deformation method

Data:
Result: Ψ
begin
  $\Psi_0 := Id$
  $n_a :=$ ComputeNumberOfAdaptSteps(f, g_0, γ_0)
  $S :=$ ComputeBlendingParameters(f, g_0, γ_0)
  for $i = 1$ *to* n_a **do**
    $\Psi_i :=$ BDM$(f_{S(i)}, g_{i-1}) \circ \Psi_{i-1}$
    $g_i := f_{S(i)}$
  $\Psi := \Psi_{n_a}$

5.7 Non-PDE based Mesh Deformation

As an alternative to the PDE-based mesh adaptation we would like to introduce a way of adapting the mesh to a geometry without having to solver PDEs. Our mesh adaptation technique that is used to achieve this increased resolution around immersed geometries is based on laplacian smoothing. Laplacian smoothing in its basic form is represented by the following equation:

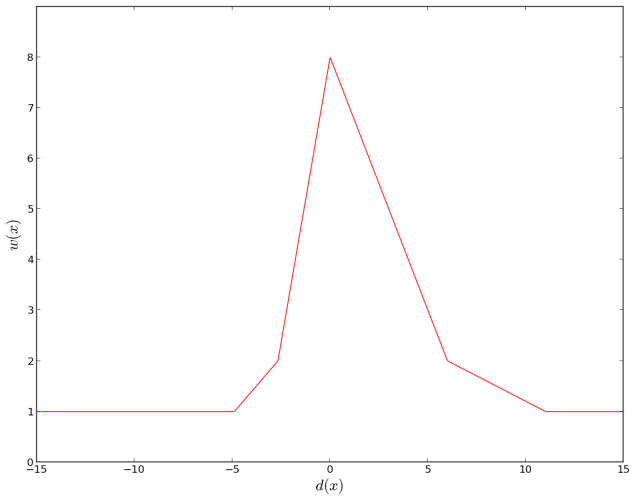
$$\mathbf{x}_{new} = (1-\omega)\cdot\mathbf{x} + \frac{\omega}{\mathrm{e}(\mathbf{x})}\cdot\sum_{j=1}^{\mathrm{e}(\mathbf{x})}\mathbf{x}_j, \tag{5.26}$$

where $\mathrm{e}(\mathbf{x})$ is the number of edges connected to the vertex $\mathbf{x}$. This can be modified into an adaptation procedure by introducing special weights that cause vertices in a certain region around the surface of the geometry to be pulled closer to the surface. The mesh adaptation formulation of the laplace smoother then reads:

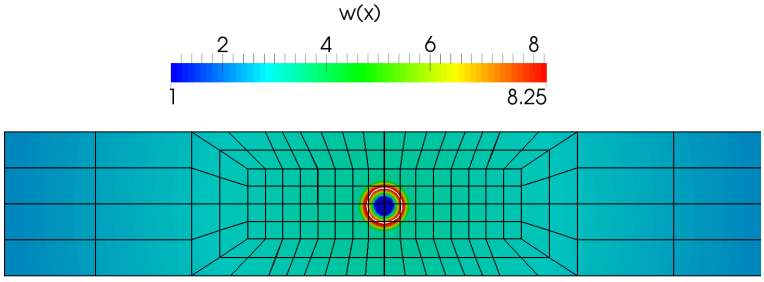
$$\mathbf{x}_i^{\mathrm{new}} = (1-\omega)\cdot\mathbf{x}_i^{\mathrm{old}} + \omega\cdot\frac{\sum\limits_{j=1,n}^{\mathrm{e}(i)} w_j\mathbf{x}_j^{\mathrm{old}}}{\sum\limits_{j=1}^{\mathrm{e}(i)} w_j} \quad \text{for} \quad i = 1, n \tag{5.27}$$

where $w(\mathbf{x})$ is the weight distribution function which in the basic formulation 5.26 is equal to 1 for all nodes. The key concept of how a laplacian smoother can be used for mesh adaptation is modification of the weight distribution $w(\mathbf{x})$ by combining it with a so called monitor function $\mathcal{M}(\mathbf{x})$. The monitor function $\mathcal{M}(\mathbf{x})$ is specific to the problem and composed with specific deformation goals in mind. For hydrodynamic force computation these goals are usually to increase the mesh resolution locally around the surface of a particle, but to leave mesh elements of the computational domain that are far away from the particle relatively unchanged. This deformation goal makes the signed distance $d(\mathbf{x})$ of the mesh nodes to the particle surface a good choice for $\mathcal{M}(\mathbf{x})$.

The monitor function can then be further modified in order to amplify the weight of mesh nodes near the particle surface and decrease the weight gradually as the distance from the particle increases until the standard weight of 1 is applied at a certain distance from the particle. To construct such a function the signed distance $d(\mathbf{x})$ needs to be combined with hat-function or similar function such as depicted in figure 5.5, the result of combining $d(\mathbf{x})$ with $\mathcal{M}(\mathbf{x})$ is then set as the weight distribution $w(\mathbf{x})$. This kind mesh adaptation by weighted laplacian smoothing corresponds the afore-



(a) The weights for the laplace smoother are generated by combining the signed distance $d(\mathbf{x})$ with a hat function



(b) Visualization of $w(\mathbf{x})$ on the undeformed mesh

Figure 5.5: Generation of the weight distribution function $w(\mathbf{x})$

mentioned laplace-κ approach, the procedure how to perform this kind of adaptation is shown in algorithm 12. The laplace-α mesh adaptation technique differs from this approach by a projection step of mesh nodes directly onto the particle surface that is

executed on the coarse mesh level (see figure 5.6) and then prolongated to the finest mesh level where in each refinement step the projection is done for new nodes that are generated from edges from the preceding mesh level. Additionally, in each refinement step a few iterations of weighted smoothing are applied. With the laplace-α approach

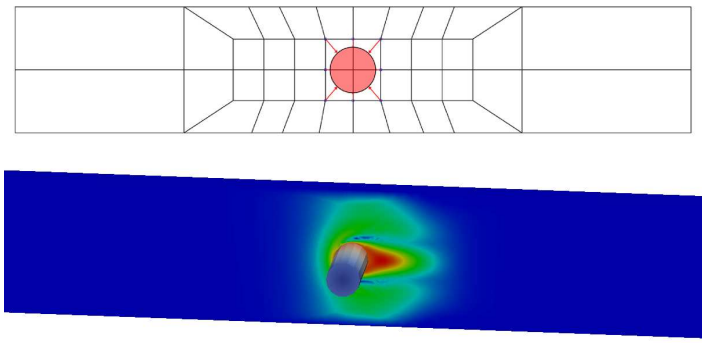
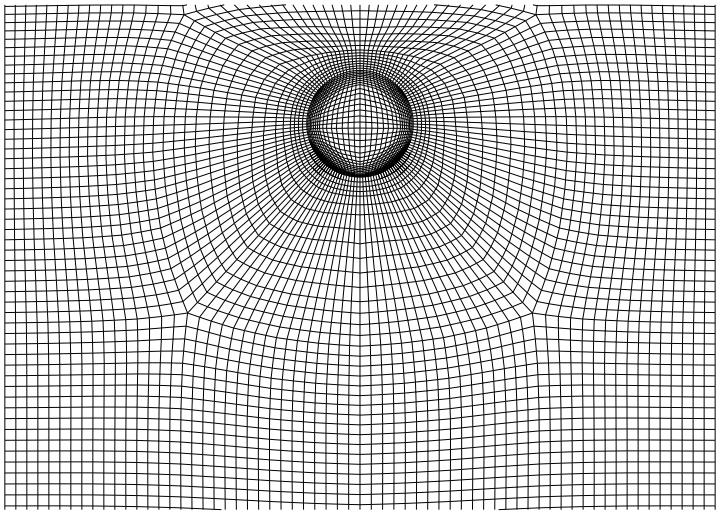


Figure 5.6: Projection of coarse grid nodes onto the surface of the object

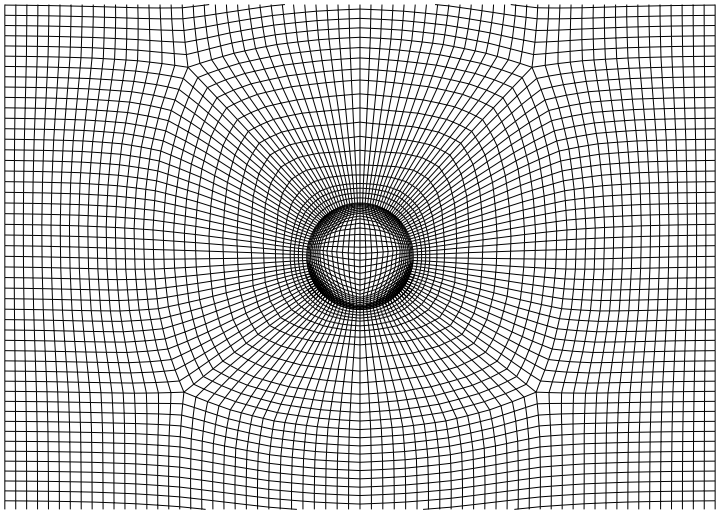
highly precise and smooth results for the hydrodynamic forces can be achieved, but also some challenges exist. The nodes that are projected to the surface of a particle follow the movement of the particle which is the key to the accuracy of the method, on the other hand this behavior becomes problematic if the particle moves longer distances as the quality mesh elements can begin to deteriorate as they get distorted by the particle movement (see figure 5.7). A strategy to avoid this kind of mesh quality deterioration with the laplace-α method is to dynamically detect this effect by evaluating mesh quality measures and in case of the mesh quality is considered not good enough to proceed the solution will be projected onto the undeformed mesh by accurate projection techniques like the $L2$ projection and then align a new set of mesh nodes to the particle surface that would produce mesh elements of better quality.

Algorithm 12: Non-PDE based Adaptation

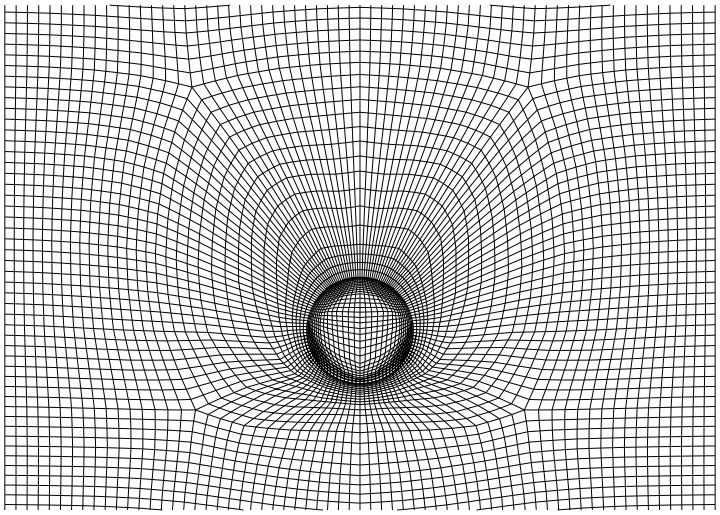
Data: Mesh m, int $stepsOnLevel[]$
Result: $mesh.coords$
begin
  for $l = 1$ *to* $m.level_{max}$ **do**
    for $i = 1$ *to* $stepsOnLevel[l]$ **do**
      $distField$ = computeDistanceField(m,l)
      n = verticesOnLevel(l)
      for $j = 1$ *to* n **do**
        $x_{j,old}$ = m.coords.getVertexOnLevel(j,l)
        m.newCoords[j] = laplaceKappa($x_{j,old}$,$distField$,$mesh$)
    prolongateCoordinates(m,m.newCoords)
    updateCoordsOnLevel(m.coords,m.newCoords,l)



(a) Laplace-α technique $t = t_0$: ball starts to fall



(b) Laplace-α technique $t = t_1$: the fixed nodes on the surface follow the ball



(c) Laplace-α technique $t = t_2$: elements in front of the ball start to get distorted

Figure 5.7: A falling ball with laplace-α deformation

Chapter 6

Results

6.1 Introduction

In this section we will test the framework that we have set up in various configurations. We at first want to show the general validity of our particulate flow fictitious boundary method in well-known benchmark configurations. Our first tests deal with single particle benchmarks for terminal velocity and sedimentation velocity. The tests analyze the velocity of the particle, the kinetic energy of the particle for different particle diameters as well as for different Reynolds numbers. We will then proceed to examine the influence of mesh adaptation on the calculation of the hydrodynamic forces in a benchmark configuration for an oscillating cylinder and for particle sedimentation. Mesh adaptation is also tested for a prototypical virtual wind tunnel scenario which will demonstrate the increase of geometry resolution by the mesh when using mesh adaptation. An advanced configuration shows the swimming of a macro-scallop swimmer in a non-newtonian fluid. The swimming mechanism is based on reciprocal motion of the swimmer. Mesh adaptation is employed to increase the resolution of the swimmer and to get more accurate results for the hydrodynamic forces that act on the swimmer. The computed result of the swimmer is compared to an experiment with a real swimmer. Sedimentation examples with complex shaped particles are presented to show the possibilities of our frame work to handle complex geometries. The proposed acceleration techniques to various parts of the FEATFLOW solver are briefly analyzed to test whether they yield the expected results.

6.2 Sphere Sedimentation towards a Solid Wall

6.2.1 Definition of the Test Case

The sphere sedimentation towards a solid wall benchmark of ten Cate et al.[13] is a well-known test configuration for Liquid-Solid solvers. We used the FEATFLOW solver with the FEM-FBM to simulate this configuration and present the results in this section. At first we briefly summarize the configuration of the benchmark. The computational domain for this test case is a boxed-shaped container with the following dimensions $depth \times width \times height = 100 \times 100 \times 160$ mm. In this domain a spherical particle is placed with the initial position of its center located at $(50, 50, 127.5)$. The diameter of the spherical particle is $d_p = 15$ mm and the density is $\rho_p = 1120$ kg/m^3. The physical properties of the fluid such as the fluid density ρ_f and viscosity ν_f are varied in order to test the solver at different Reynolds numbers. The experiments performed by ten Cate focus on four main test cases which are referred to as E1-E4. The corresponding simulations are called S1-S4. In table (Tab. 1) the different configurations are summarized, where u_∞ denotes the maximum sedimentation velocity in a container of infinite height and Re refers to the Reynolds number, calculated as $Re = d_p u_\infty \rho_f / \nu_f$. The simulations S1-S4 are carried out over a total simulation

Table 6.1: Fluid properties for the different test cases

Case nr	ρ_f [kg/m^3]	ν_f [Ns/m^2]	u_∞ [m/s]	Re [-]
E1	970	0.373	0.038	1.5
E2	965	0.212	0.06	4.1
E3	962	0.113	0.091	11.6
E4	960	0.058	0.128	31.9

time of 5.0 seconds. We store various data for each case. Most importantly we store the velocity of the particle up, the height of the particle h, the kinetic energy of the fluid domain $E_{kin,f}$ and the kinetic energy of the spherical particle $E_{kin,p}$. To calculate $E_{kin,p}$ we use the standard kinetic energy equation for rigid bodies:

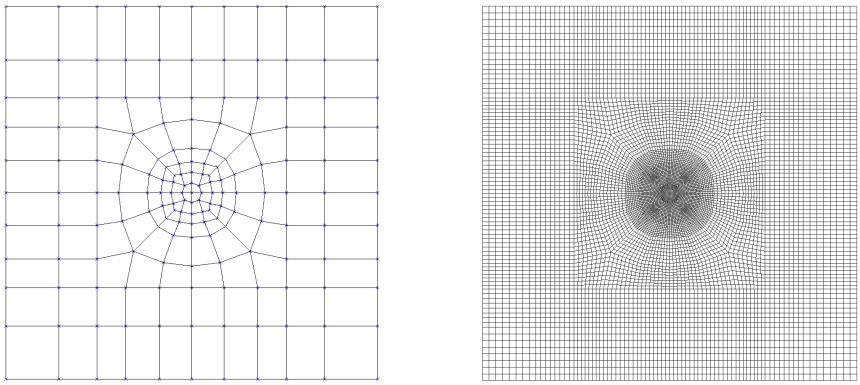
$$E_{kin,p} = \frac{1}{2} m u^2 \tag{6.1}$$

The kinetic energy of the fluid is defined as:

$$E_{kin,f} = \int_\Omega \rho_f u^2 d\Omega \tag{6.2}$$

In a finite element framework this can be efficiently evaluated as:

$$E_{kin,f} = U^T M_\rho U \tag{6.3}$$



(a) The coarse mesh is pre-adapted to the particle along the z-axis

(b) The mesh resolution increases especially in the particle region

Figure 6.1: Configuration of the computational mesh for the benchmark

6.2.2 Simulation Results

In this section we present the results of the simulations S1-S4 using the multigrid FEM-FBM. The computational domain was discretized using an unstructured mesh T with different levels of refinement. The coarse mesh is constructed in a way that the mesh resolution along the expected trajectory of the particle is greatly increased. A x-y-slice of the mesh is shown in figure 6.1a. The complete 3D mesh is then generated by this mesh element using extrusion. The coarse grid T1 consists of 2100 vertices and 1216 hexahedral elements, we computed the results on refinement level 4 which corresponds to grid T4 with 1075200 vertices 622592 elements, a x-y-slice of the fine mesh is shown in figure 6.1b to illustrate the resulting mesh resolution. The simulations were run for a total simulation time of 5.0 seconds using a time step size $\Delta t = 0.001$. At first we examine the sedimentation velocity observed in the simulations S1- S4 and then compare this data to the experimental and simulation data provided by ten Cate. The sedimentation velocity is shown in figure 6.2 on the maximum mesh resolution level T4. To compare the simulation data with the data obtained in the experiment ten Cate used the ratio of the maximum velocity measured in the experiment to the theoretically determined maximum sedimentation velocity and the ratio of the maximum velocity in the simulation to the theoretical maximum. We computed these values and summarized the comparison of our values to those of the experiment and simulation by ten Cate in table 6.2.

We analyzed particle position data as well and plotted the particle trajectories in figure 6.3. The particle trajectory is measured in the dimensionless height of the gap between the particle and the bottom wall of the domain h/d_p. The values obtained for the sedimentation velocity and particle trajectories compare very well to those of the experiment and the simulations of ten Cate in all of test cases S1-S4. Characteristic features of the particle behaviour like the velocity plateau that is reached in the S1 case or the relatively abrupt deceleration of the particle in the S4 case are clearly visible.

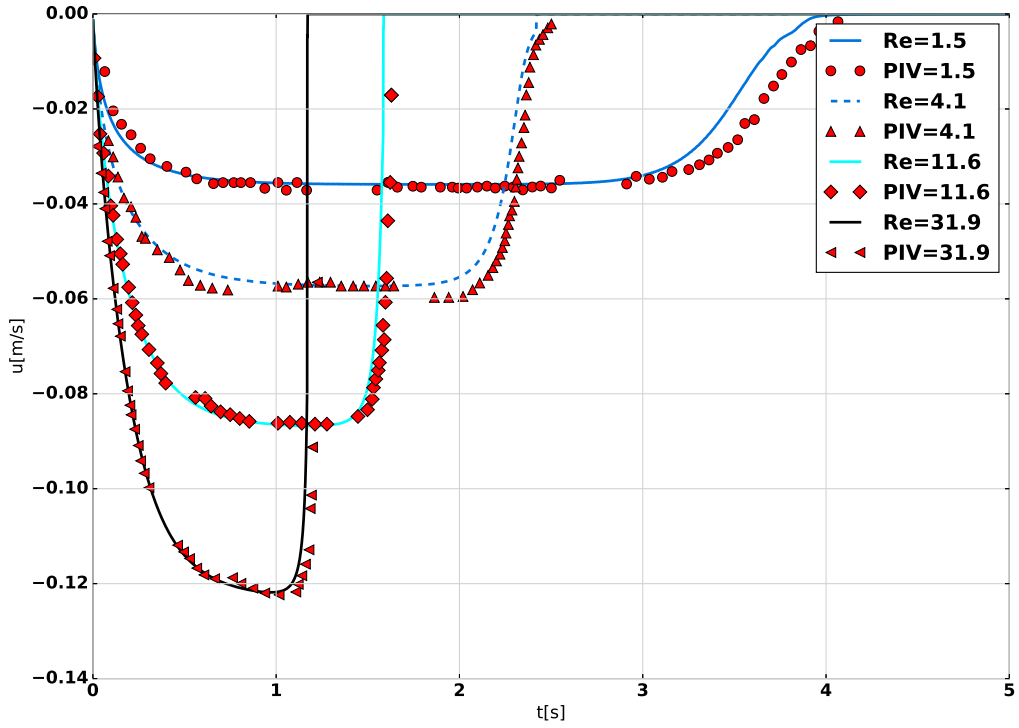


Figure 6.2: Sedimentation velocity at different Reynolds numbers.

Furthermore, the method is able to capture flow features like the deceleration phase of the particle in cases S1-S3 when it close to the bottom wall of the domain without the introduction of additional forces on a sub-mesh level. We attribute this to the high mesh resolution that the method is able to handle efficiently.

Influence of Mesh Refinement

To demonstrate the validity of our multigrid framework, we computed the S3 and S4 cases on different levels of mesh refinement. The underlying assumption is that the accuracy of the results should improve in case that the mesh refinement level is increased. In figure 6.4 (for case S4) we display the sedimentation velocity computed on different refinement levels. In order to evaluate the result a line indicating the maximum velocity measured in the experiment is inserted.
We can observe that as the mesh refinement increases the maximum velocity reached in the simulation gets closer to that of the experiment in both of the analyzed cases. Additionally, the shape of the curves seems to get closer to those published by ten Cate. This behaviour is a strong indication that the underlying multigrid framework is working as expected.

Table 6.2: Fluid and particle properties

Re	u_{max}/u_∞	u_{max}/u_∞	u_{max}/u_∞
	Featflow	ten Cate	Experiment
1.5	0.945	0.894	0.947
4.1	0.955	0.950	0.953
11.6	0.953	0.955	0.959
31.9	0.951	0.947	0.955

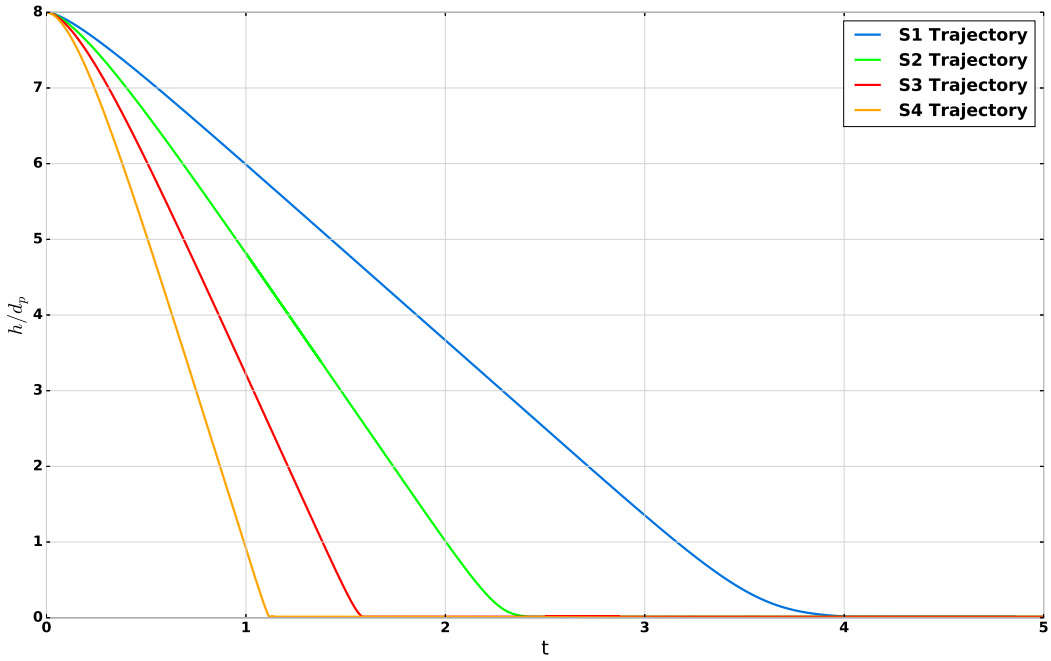


Figure 6.3: Particle Trajectory for the different test cases

6.2.3 Comparison with other CFD-Codes

We include a comparison of our results for the sedimentation velocity (figure 6.5) and the kinetic energy (figures 6.6,6.7) with simulation results from the research group of Sommerfeld and Ernst which were produced by a Lattice-Boltzmann code, as well as the original simulation results of ten Cate.

Particle and Fluid Kinetic Energy

The data calculated for the kinetic energy of the fluid and the particle also closely matches the experimental data and the simulations of ten Cate. For the test cases S2 and S3 we do not have the original results from ten Cate available, so for these cases

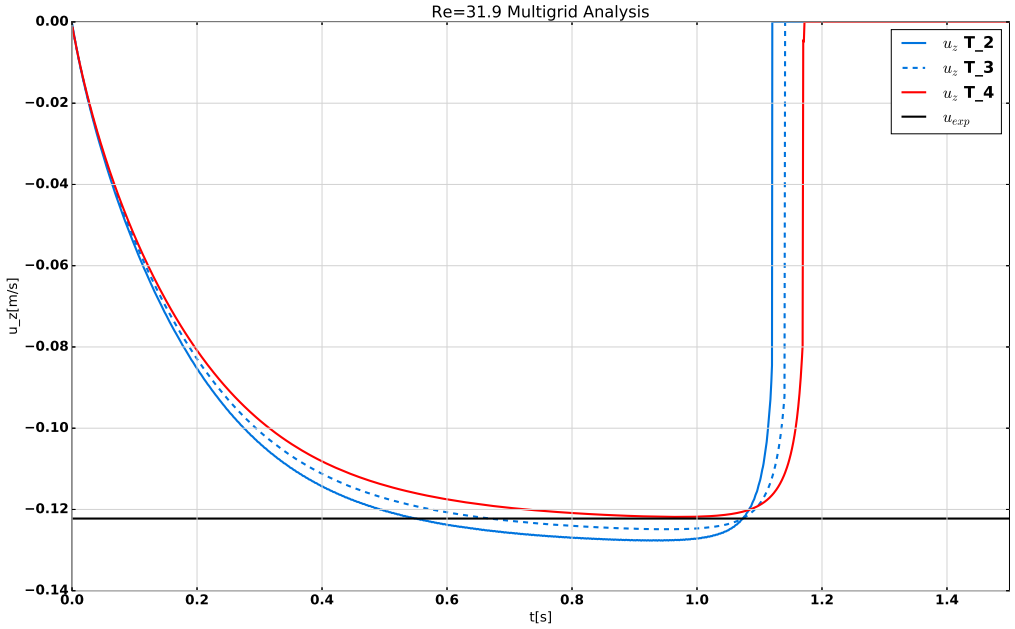


Figure 6.4: Sedimentation velocity evolution for different mesh refinement levels

we only show the comparison between our results and those of the Lattice-Boltzmann calculations by Sommerfeld.

Conclusions of the Sphere Sedimentation Benchmark

The results show that our method is able to handle the sphere sedimentation case well and produce results that compare very well to the PIV readings of the original experiments as well as to simulation results from the group of ten Cate and from the group of Sommerfeld you employ a Lattice-Boltzmann approach. The multigrid framework seems to work as expected and we already see accurate results for the benchmark at a refinement level that is two levels below the maximum level used in our computations. Regarding the kinetic energy plots we observe that the simulation results of ten Cate differ from both our results and the results of the Sommerfeld group. We suppose that the results of newer simulation from our side and that of the Sommerfeld group were able to resolve the domain better and thus more accurately, which might suggest that these newer results are actually an improvement on the original calculation. If we observe that other groups as well come closer to our results and those of the Sommerfeld group this would add further evidence to our suggestion.

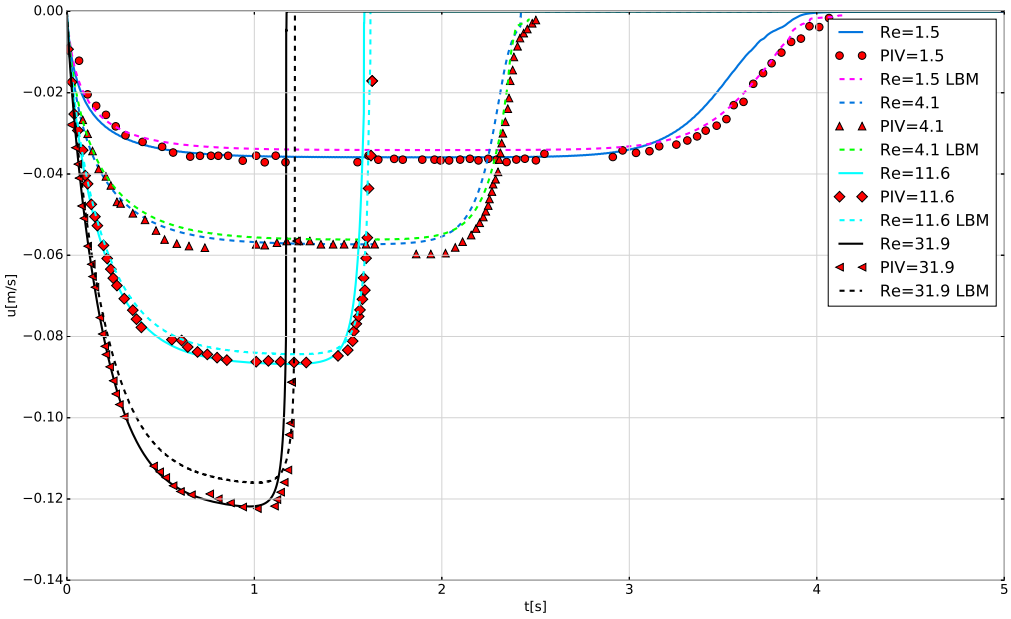
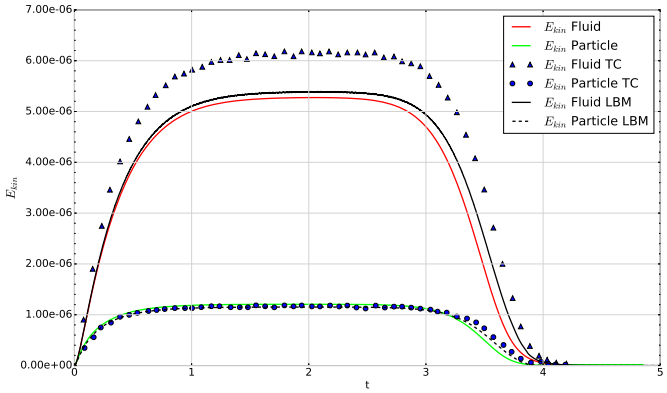
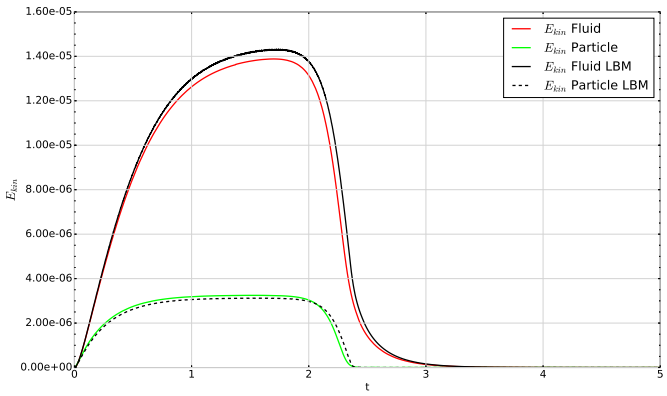


Figure 6.5: Sedimentation velocity compared to PIV readings and to LBM simulations the Sommerfeld group

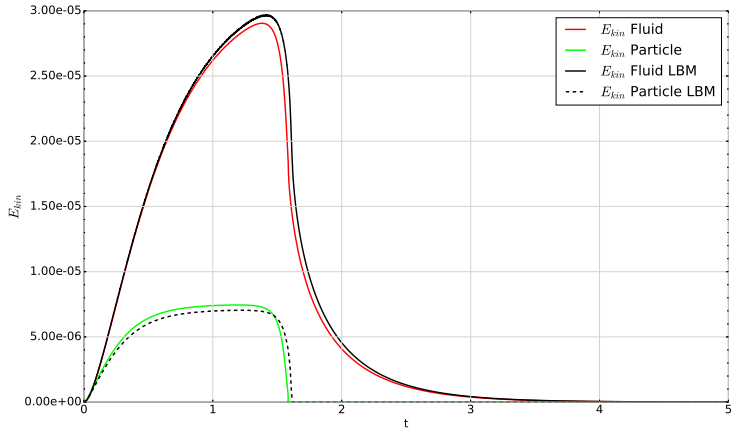


(a) Kinetic energy for the test case S1

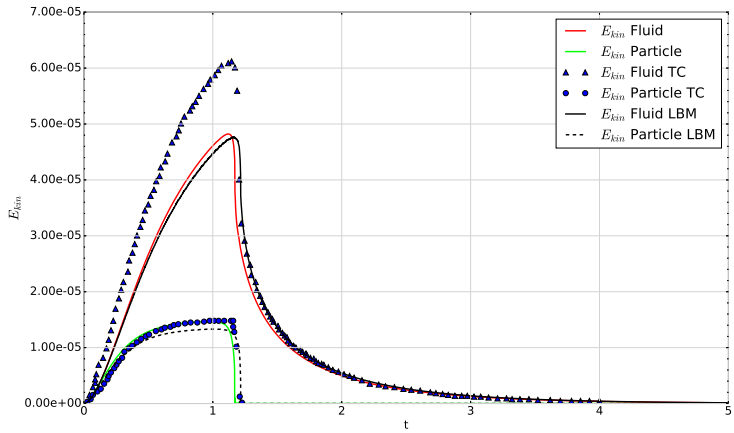


(b) Kinetic energy for the test case S2

Figure 6.6: Time evolution of the kinetic energy for the fluid and the particle (S1,S2)



(a) Kinetic energy for the test case S3



(b) Kinetic energy for the test case S4

Figure 6.7: Time evolution of the kinetic energy for the fluid and the particle (S3,S4)

6.3 Oscillating Cylinder in a Channel

This benchmark setup features an oscillating cylinder in a brick-shaped channel. The quantities of interest for this particular case are the drag and lift forces acting on the cylinder (diameter $D = 0.1$) and respectively the drag and lift coefficients measured over time. The oscillating cylinder movement is governed by a sinusoidal function that changes only the x-coordinate of the cylinder center (X_c, Y_c) whereas the y- and z-coordinates stay constant. The initial position of the cylinder $(X_0 = 1.1, Y_0 = 0.2, Z_0 = 0.1025)$ is used to calculate the position of the cylinder at time t as $(X_c(t) = X_0 + A \cdot sin(2\pi ft), Y_c(t) = Y_0, Z_c(t) = Z_0)$, where $A = 0.25$ and $f = 0.25$ are the amplitude and frequency of the oscillating position function. The size of the computational domain is $(2.2 \times 0.41 \times 0.1025)$, which means that the cylinder is not located directly in the middle of the y-range so that a non-zero lift is generated. The fluid parameters for this benchmark are given by a kinematic viscosity of $\nu = 10^{-3}m^2/s$ and a density of $\rho = 1kg/m^3$ with the fluid being at rest initially. As a reference for this benchmark we use the 2D simulations conducted by Wan, Turek and Rivkind [92], where the drag and lift were calculated on a body-aligned mesh meaning that the circle in their case was build directly into the mesh. In order to make the results comparable with 3D simulations the drag and lift coefficients (c_d, c_l) need to be scaled by the thickness of the channel and are thus calculated by:

$$c_d = \frac{2F_d}{\rho \overline{U}^2 DT}, \; c_l = \frac{2F_l}{\rho \overline{U}^2 DT} \tag{6.4}$$

where F_d and F_l are the hydrodynamic drag and lift forces, D the diameter of the cylinder and T the thickness of the domain.

6.3.1 Setup of the Benchmark

In this test case we want to compare the results of our mesh adaptation technique in comparison to a standard FBM approach where the geometry resolution is improved by regular refinement of the mesh. The quantities we will focus on in this benchmark are the hydrodynamic drag and lift forces. The geometry resolution obtained by the different methods is shown in figure 6.9 for the regular refinement approach, in figure 6.10 for the Laplace-κ approach and in figure 6.11 for the Laplace-α approach. In our simulation we will compute several cycles of the cylinders oscillating left-right motion and measure the drag and lift forces and compute c_d and c_l. The number of elements for the different mesh levels used in the computation are summarized in table 6.3. We see that the regular refinement approach is refined once more than in the case that we use mesh adaptation techniques. A 2D slice of the coarse mesh of the computation and the initial location of the cylinder are shown in figure 6.8, from this slice a 3D mesh is generated by extrusion. On these meshes we compute several cycles of the cylinders periodic motion for all the standard FBM as well as the Laplace-κ and Laplace-α approaches. In both cases with mesh adaptation the mesh is 'regularized'

using a few steps of standard Laplacian smoothing after each time step, before it is then again adapted to the new position of the cylinder. This mesh regularization provides a better base mesh for the following adaptation and usually results in a more uniform adaptation over time as opposed to the case where a strongly adapted mesh is used as the base for the next adaptation step.

Table 6.3: Number of mesh elements for different adaptation techniques

LEVEL	4	5
Standard FBM	-	1310720
Laplace-κ	181220	1310720
Laplace-α	181220	1310720

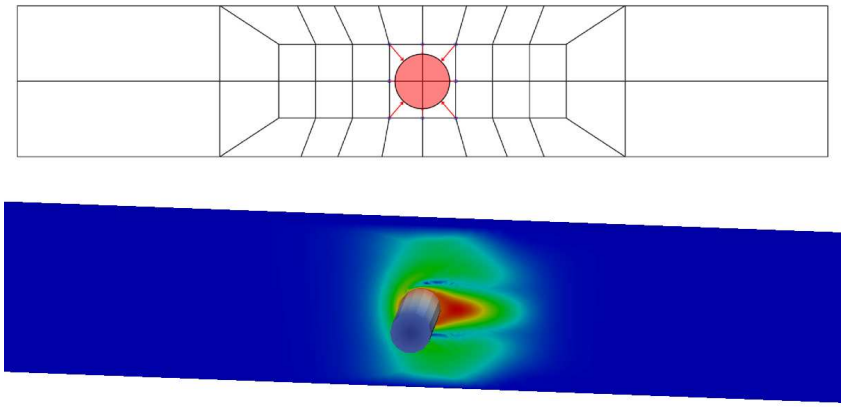
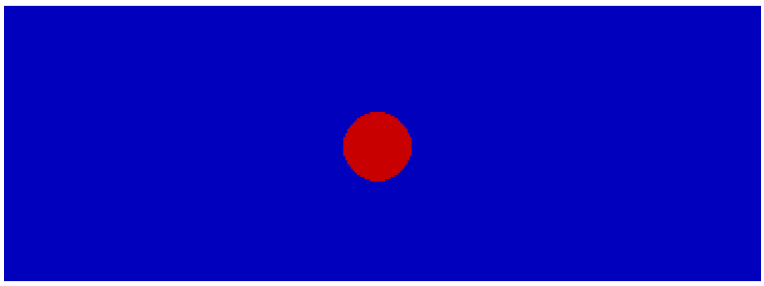


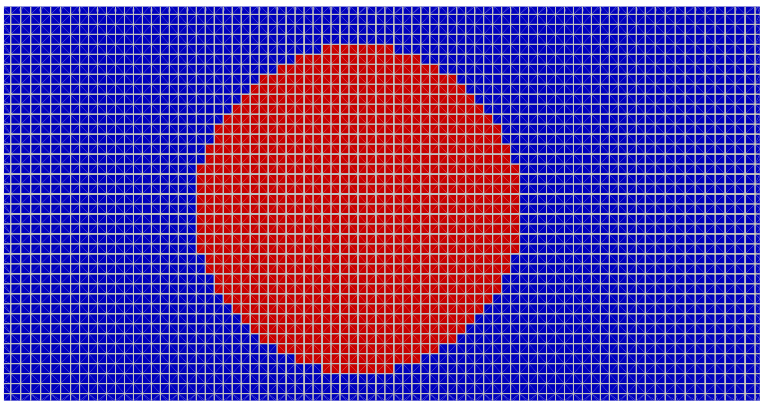
Figure 6.8: Coarse grid used for the oscillating cylinder benchmark and initial location of the cylinder

Discussion of the Achieved Geometry Resolution

We can see from figures 6.9, 6.10 and 6.11 that even though the standard FBM approach is one level higher in mesh refinement that the surface of the cylinder is not resolved as smoothly as in the cases when we use mesh adaptation. In the case of Laplace-κ adaptation we observe that the mesh vertices are located near the surface of the cylinder. Although we see that vertices are near the surface, we also see that the faces of the elements are not located exactly on the surface which would lead to an even better geometry approximation. We demonstrate in figure 6.10 by the cells colored in green the boundary cells of the surface. The actual cylinder geometry only intersects these cells in a way that the faces of the cells are not aligned with the cylinder geometry which is the difference to the second adaptation technique that we use, the Laplace-α technique. For this technique we see that by the placement of the faces



(a) Cylinder appears round, but with slightly jagged edges in a two-color map



(b) Mesh resolution around the cylinder

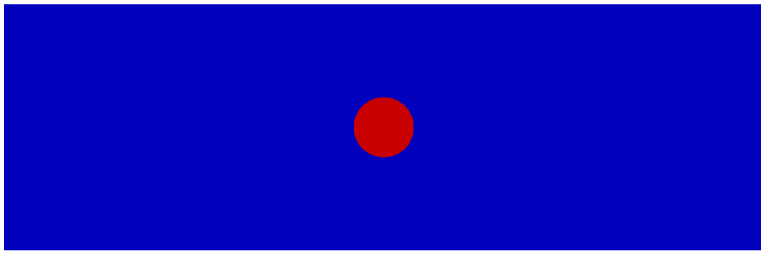
Figure 6.9: Cylinder resolution by the standard FBM regular refinement strategy

of the mesh directly onto the surface of the cylinder we achieve a perfect resolution of the cylinder.

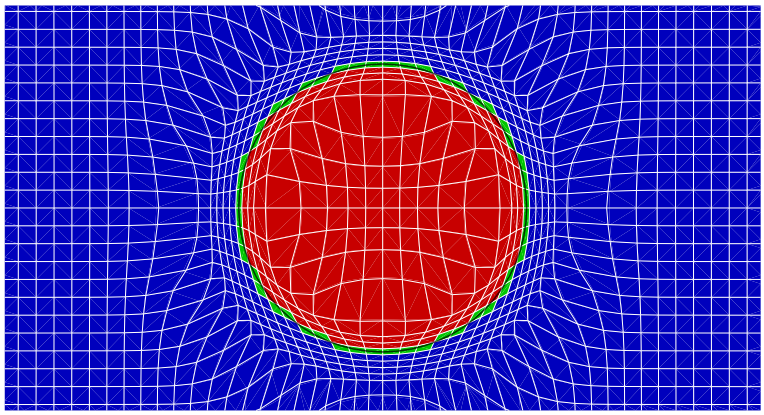
Comparison of the 3D Results for Laplace-α to the 2D Results

When we compare the results computed with the Laplace-α with the 2D results of Wan et al. we can observe over the course of all computed cycles excellent argreement and a highly smooth curve of the development of the drag coefficient c_d. The drag curve for Laplace-κ shows some oscillations in comparisons to the Laplace-α method, but it still agrees with the 2D very well. The oscillations occur at the turning point of the cylinders motion from negative x-direction to positive x-direction.

For the lift results we see the trend observed for the drag continue: the curve for the Laplace-α technique matches the 2D results excellently. The Laplace-κ approach displays more oscillations than for the drag, but it still agrees very well with the results from Wan and those of the Laplace-α calculations. The increase in oscillations com-



(a) Cylinder appears round in a two-color map



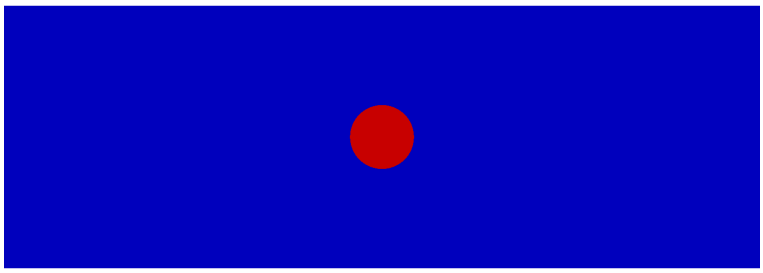
(b) The mesh nodes are concentrated near the surface of the cylinder

Figure 6.10: Cylinder resolution for the Laplace-κ adaptation

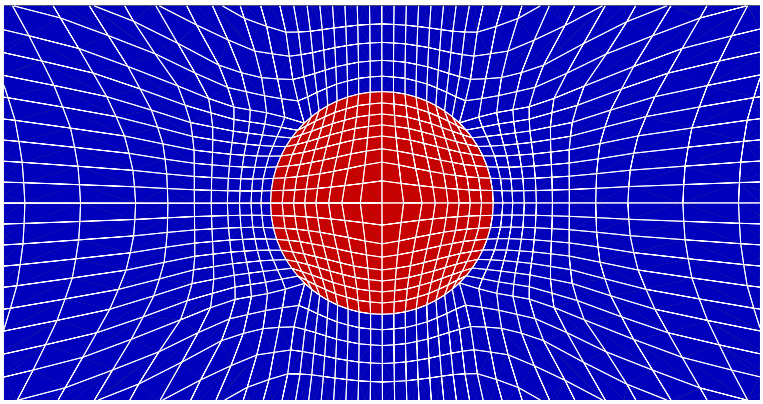
pared to the drag computation is to be expected because the lift value in this case is a numerically much smaller number which can increase the strength of the oscillations.

Comparison with the standard FBM Approach

When we compare the standard FBM approach to the results computed with mesh adaptation techniques (figure 6.14) we can see a good agreement in the drag curves in all computed cycles in the ascending and descending parts of the curve. At the peak and low points of the curve we can see more and higher amplitude oscillations than for the Laplace-κ and Laplace-α results. Additionally, the standard FBM curve does not match the values of the Laplace-κ and Laplace-α computations at the low and high points of the curve. For the lift curves (see figure 6.15) we see a significant difference in the amplitude of the oscillations for the standard FBM approach compared to the Laplace-κ as well as the Laplace-α lift curves. But not only do we observe an increase in oscillations, we as well can see that although the general shape of the lift curve for the standard FBM approach matches that of the Laplace-α approach the standard FBM



(a) Cylinder appears round in a two-color map



(b) The mesh nodes are located exactely on the surface of the cylinder

Figure 6.11: Cylinder resolution for the Laplace-α adaptation

tends to overshoot some values which leads to larger values in the extreme points of the lift curve.

Comparison of C_d and C_l for Laplace-α and Laplace-κ Adaptation

When looking at the differences for the Laplace-α and Laplace-κ curves we that they differ the most in the stage shortly after the sign of the cylinder velocity is changing from positive to negative and vice versa (see figure 6.16). This can be explained by the fact that when the motion of the cylinder changes some of the mesh nodes change their status from solid to fluid abruptly or the nodes change from positive velocity to negative velocity from one time step to the other which leads to the oscillations observed expecially in the case of the standard FBM where these situations arise far more often than in the case of the Laplace-κ approach. The Laplace-α approach does not suffer from these problems because the nodes never change from fluid to solid.

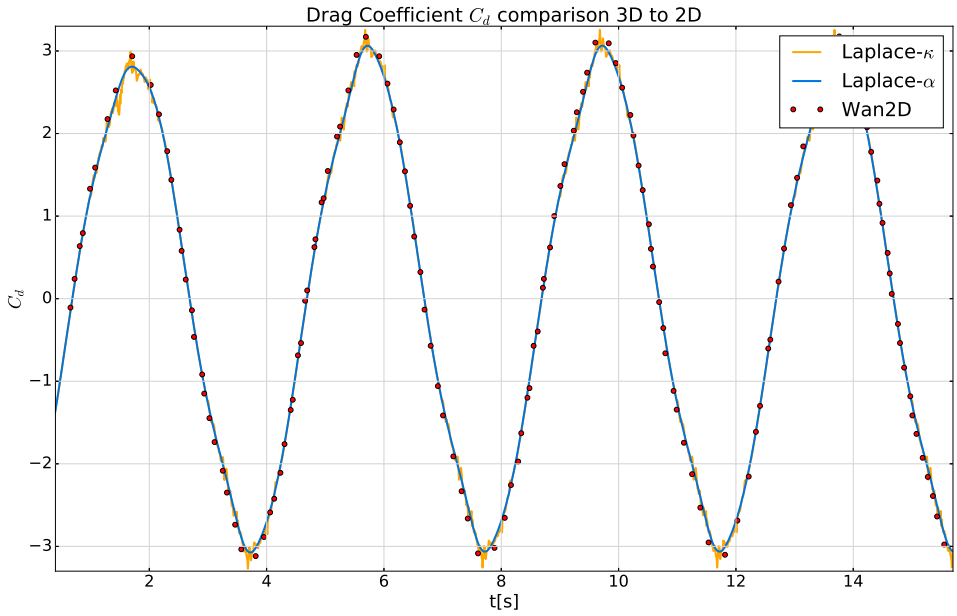


Figure 6.12: Drag coefficient comparision between Wan's and our results.

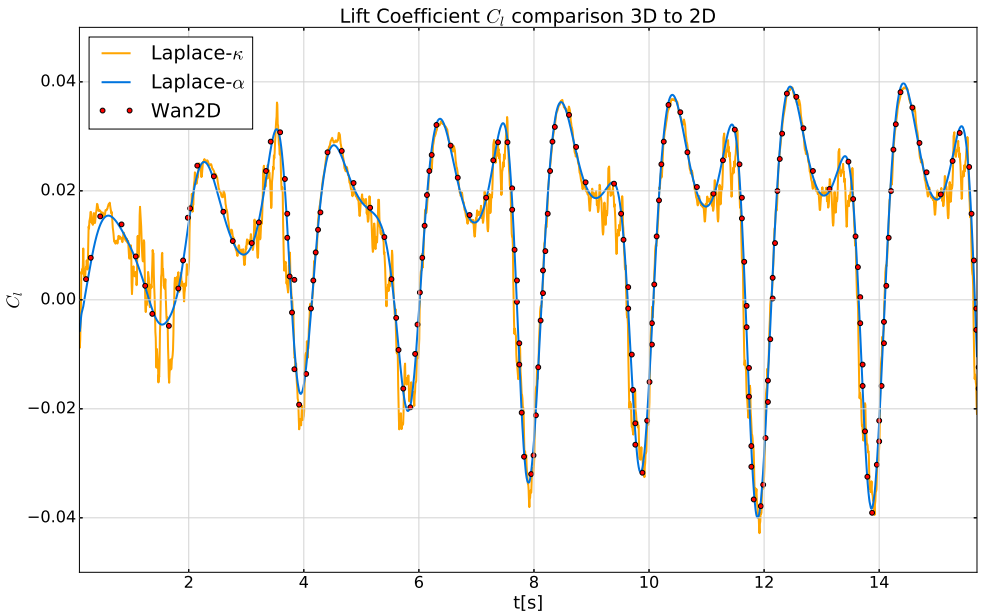


Figure 6.13: Lift coefficient comparision between Wan's and our results.

Conclusions of the Benchmark

We can safely conclude that both mesh adaptation techniques Laplace-α and Laplace-κ offer a significant improvement over the standard FBM approach with regard to the computation of the hydrodynamic forces. The force curves are clearly smoother which will lead to better results during the numerical integration of the equations of motion when particles are allowed to move freely. Also the geometry resolution is clearly superior with mesh adaptation and as the results in figures 6.9, 6.10 and 6.11 demonstrate the effect of mesh adaptation can be considered more than one level of regular refinement for the case at hand. The convergence test shown in figure 6.17 confirms that we are dealing with converged results.

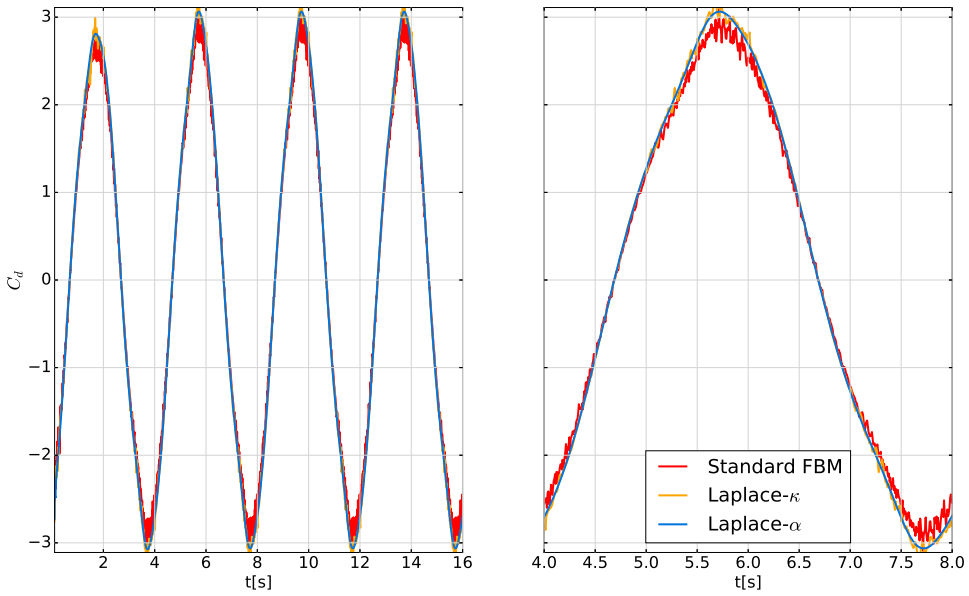


Figure 6.14: Drag curves for the standard FBM, Laplace-α and Laplace-κ calculations

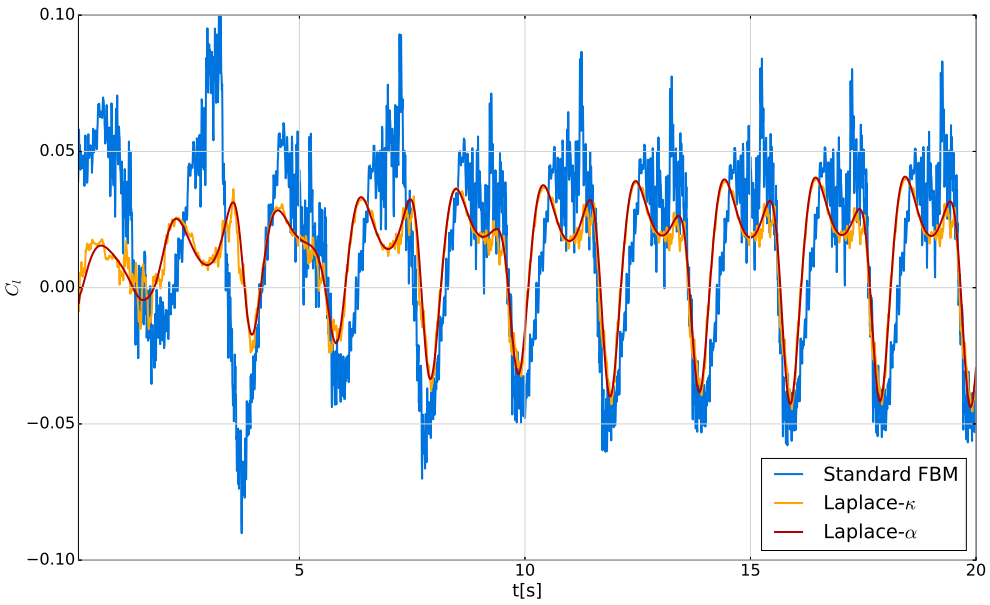


Figure 6.15: Drag curves for the standard FBM, Laplace-α and Laplace-κ calculations

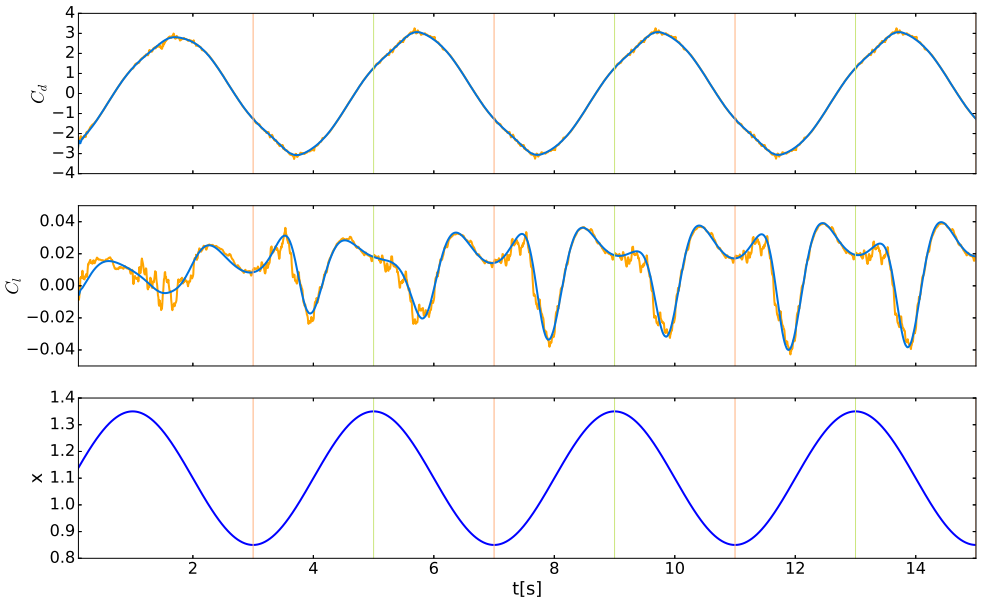


Figure 6.16: Drag and lift curves combined with a position plot of the cylinder

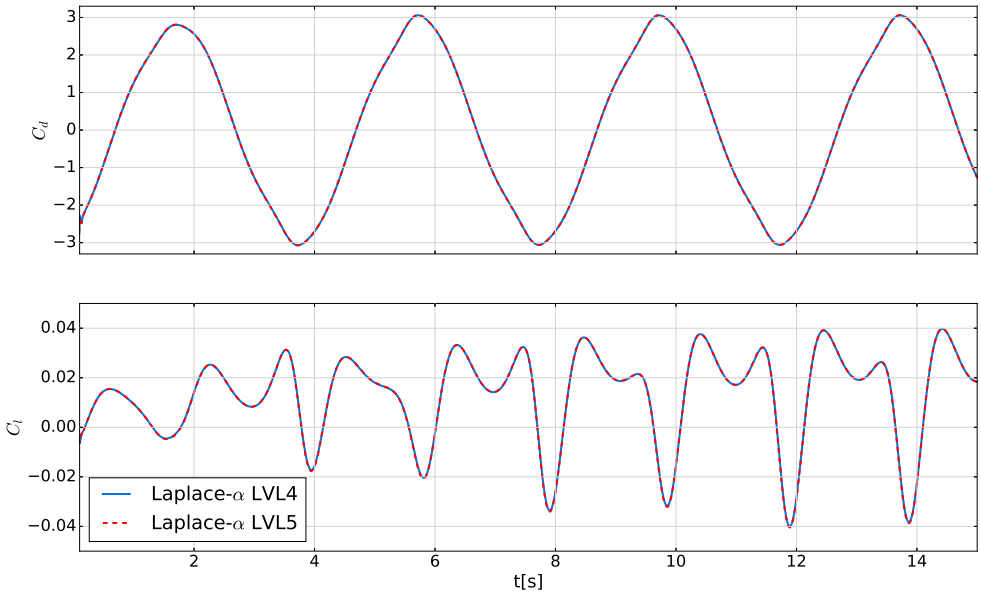


Figure 6.17: Laplace-α results for two grid levels, showing converged calculations

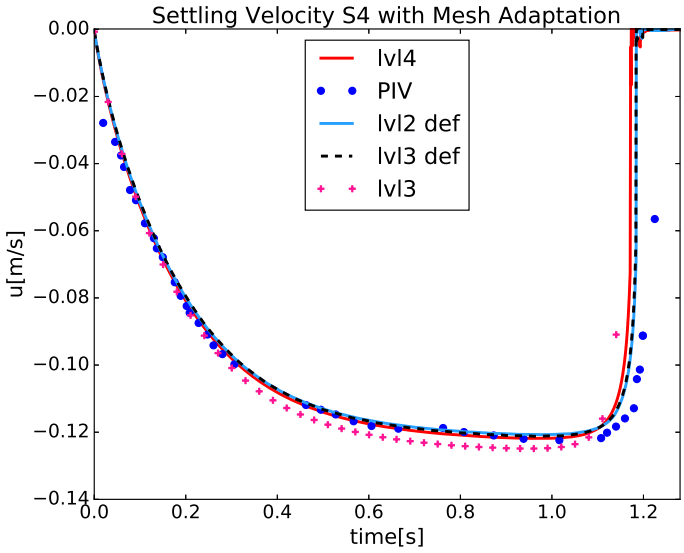
6.4 Sphere Sedimentation with Mesh Adaptation

We want to test the influence of mesh adaptation on the sphere sedimentation benchmark. We choose the Laplace-κ mesh adaptation technique as it is more general and more easily applied, while the Laplace-α technique might yield better results we might run into the problems depicted in section 5.7. In the case of the oscillating cylinder we saw by using mesh adaptation the differences in results in comparison to regular refinement can be rated as about one level of refinement. It has to be kept in mind that in the last case even if the curves for drag and lift showed more oscillations without mesh deformation, the average values corresponded closely to the results with mesh deformation. In sedimentation the force values are integrated over time and we arrive at the particle velocity, we suggest that also for this quantity the results should improve with mesh deformation, but since the integration itself is an averaging process the differences might not be as significant as in the oscillating cylinder case. For these tests we use a different mesh because the mesh for the sphere sedimentation test was already extremely refined in the region of the particle by certain connector elements that allow for a transition from larger cells on near the boundary and in the flow region to smaller cells in the region near the particle and in the assumed trajectory of the particle (see figure 6.1a). The number of vertices and elements for the meshes used in shown in figure 6.4

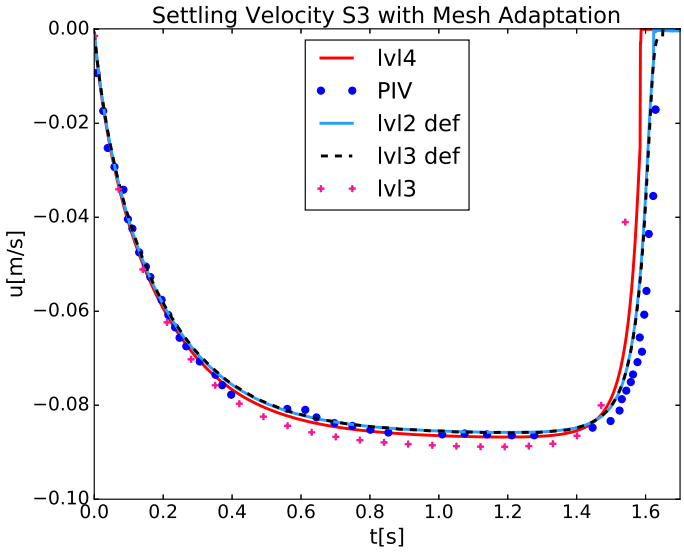
The results of our computations with mesh adaptation are shown in figure 6.18 com-

Table 6.4: Number of mesh elements and vertices for sedimentation test

LEVEL	2	3	4
Vertices	112128	897024	3823696
Elements	131956	1310720	3588096



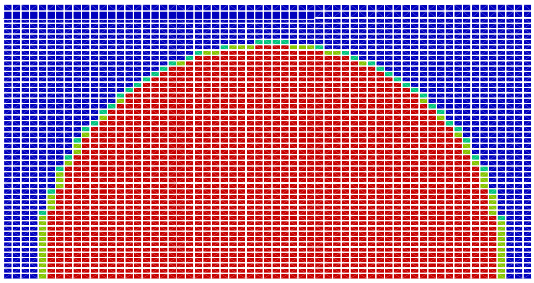
(a) Settling velocity for S4 with mesh adaptation



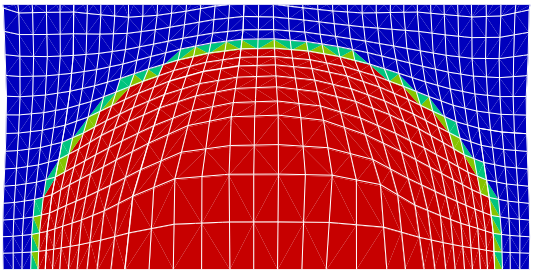
(b) Settling velocity for S3 with mesh adaptation

Figure 6.18: Comparing settling velocity for regular refinement with mesh adaptation

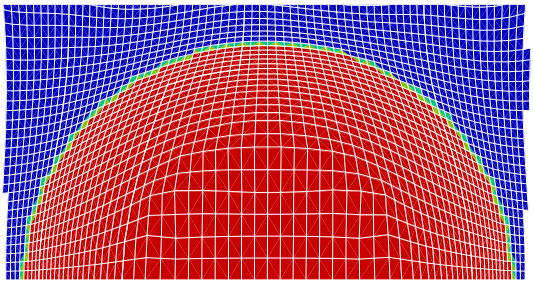
pared to the results for the regular refinement FBM approach. We see that the LVL2 as well as the LVL3 computations closely match the solution of the regular refinement approach at LVL4. The LVL3 curve is however only a slight improvement to the LVL2 solution curve. Furthermore, we see that both the LVL2 and the LVL3 curves capture the deceleration phase of the particle better than the LVL4 curve. For the deceleration phase the results with mesh adaptation are clearly superior to the calculation without it. We see that these observations hold true for both test cases S3 and S4, so there is consistency of the beneficial behavior of mesh adaptation. The resolution of the sphere surface for a slice of the mesh achieved by the different approaches is shown in figure 6.19. We see that although the LVL4 regular refinement calculation has a much higher number of elements the shape of the sphere is still not as smooth as in the cases with mesh adaptation and we can still see the sharp jagged edges of the mesh. The small improvement between the LVL2 and LVL3 calculations with mesh adaptation we attribute to the already well resolved geometry shape that is achieved on LVL2. The LVL3 results with regular refinement are inferior to the LVL2 and LVL3 calculations with mesh adaptation in all aspects.



(a) Regular refinement



(b) Laplace-κ LVL2



(c) Laplace-κ LVL3

Figure 6.19: Sphere geometry resolution for a slice of the mesh

6.5 Numerical Simulation of Swimming at Low Reynolds Numbers

6.5.1 Introduction of the Test Case

The analysis of swimming mechanisms of biological microswimmers is a current topic in biology, physics and engineering. The aim of such studies is to understand the underlying mechanisms and to apply them to synthetic microswimmers that can be used for special purposes in a wide range of biomedical or engineering applications. The swimming of biological swimmers is governed by Purcell's scallop theorem [67] as a consequence of which they generally execute non-reciprocal motions in low Reynolds number fluids in order to propel themselves forward. The theorem furthermore states that if a swimmer applies a geometrically reciprocal swimming mechanism, that is a

motion which is identical when reversed, then the net displacement of such a swimmer has to be zero. In case of non-Newtonian fluids, which are present in most biomedical applications, the scallop theorem does not apply anymore. A swimming mechanism that is based on reciprocal motion should be possible in these fluids. Such a swimmer is simulated in this section using the FBM technique, the results are compared to experimental data for the same configuration as well as to some theoretical considerations. The real swimmer that was used in the experiment is called a macro-scallop with a total length of $l = 22mm$. The real macro-scallop has a head in which two motors are embedded that power the motion of the macro-scallop's shells. The shells are made of carbon fiber sheets that are $0.3mm$ thick and $16mm \times 14mm$ in length and width. The schematics of real the macro-scallop swimmer are depicted in figure 6.20. Further information on the configuration of this simulation can be found in [68]. A full cycle of the opening/closing motion of the swimmer is configured to be $4s$. In

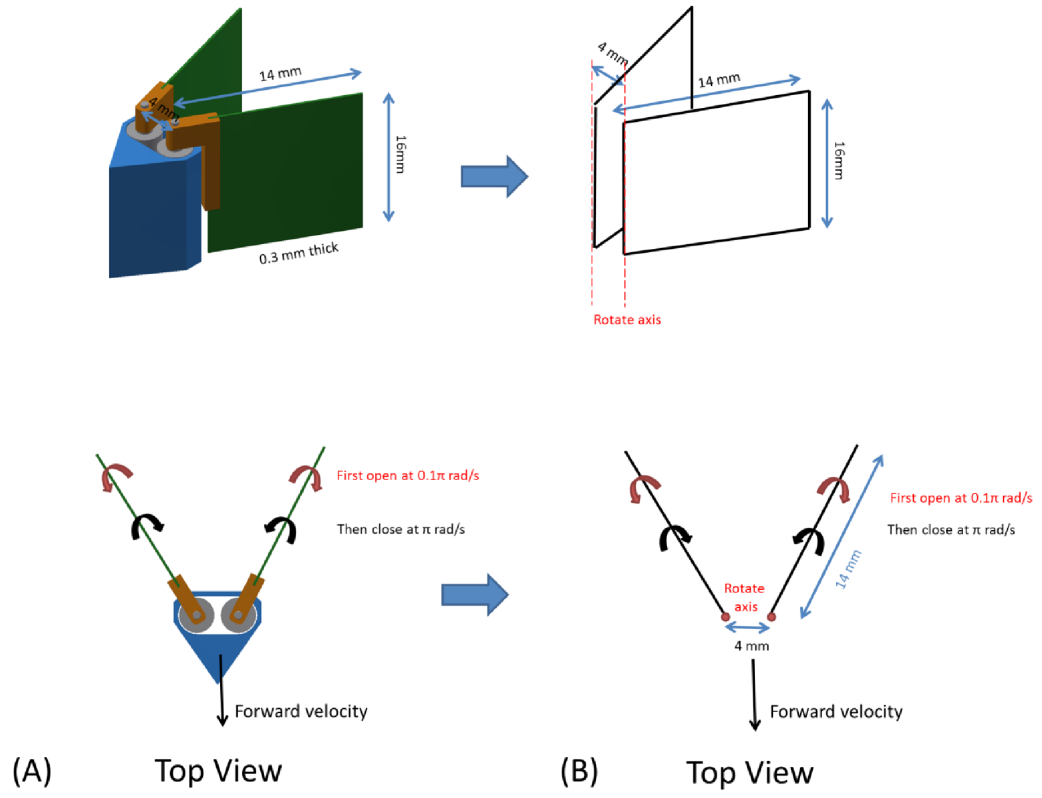


Figure 6.20: Schematics of the macro-scallop and illustration of the reciprocal shell motion.

theory [68] such a swimmer is expected to exhibit a positive *net displacement* D/l, where D is the total forward displacement (see figure 6.21) of the swimmer and l the length of the swimmer, in case the closing speed of the shells is higher than the open-

ing speed: $\omega_c > \omega_o$ and the surrounding fluid is non-Newtonian. The value of the net displacement is also dependent on the relation $p = \frac{\omega_c}{\omega_o}$ and the angles α_o, α_c that are formed between the shells of the swimmer at the opened and at the closed position. For the swimmer in the experiment these angles are set to $\alpha_o = 237°$ and $\alpha_c = 10°$. As a control experiment the macro-scallop swimmer was tested in silicone oil which can be classified as a highly viscous Newtonian fluid. The precise description of the silicone oil is: (Dow Corning 200/12,500 cSt, VWR, UK).

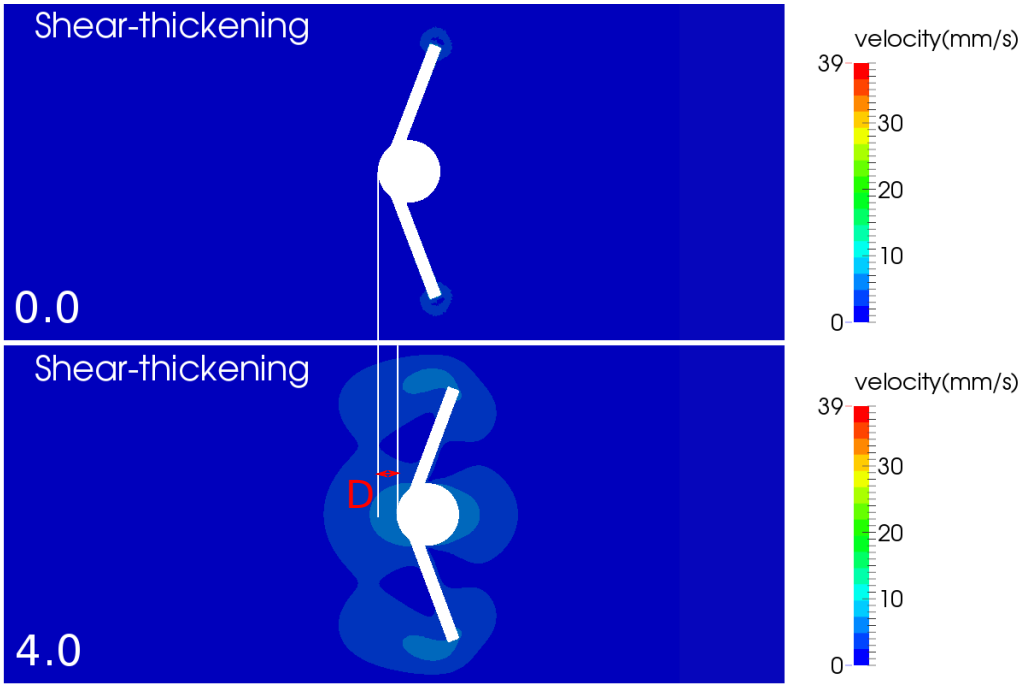
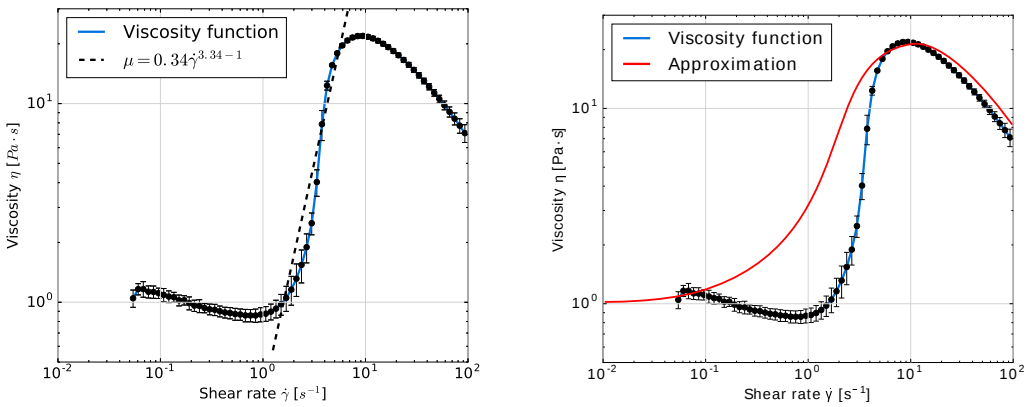


Figure 6.21: A single cycle of the macro-scallop is $4s$ long. We sketch the forward displacement D from which we calculate D/l.

The shear thickening fluid was produced by fumed silica suspensions (8% w/w). The Fumed silica powder (Aerosil®150, Evonik, Germany) was mixed thoroughly with poly(propylene glycol) (PPG, Mw=725, Sigma-Aldrich) and the resulting solution was then degassed for three hours. Then dynamic viscosity measurements were performed which can be used to establish a viscosity function. The viscosity function of the shear-thickening fluid that was obtained by the measurements is shown in figure 6.22a. We can see the viscosity function shows a very steep gradient for shear rates between 3 and 9. In the simulation this steep gradient can cause the solver to diverge and in order to improve the reliability of the solver we slightly modified the viscosity function such that we left the minimum and maximum values intact, but made the change more gradually over a larger shear rate interval as is shown in figure 6.22b. As we can see the dynamic viscosity of this shear thickening fluid is in the range of $[0.9, 22]$ $Pa \cdot s$ and its density is $\rho = 1051$ $kg \cdot m^{-3}$. In the simulation the swimmer geometry was slightly



(a) Dynamic viscosity measurements from experiments

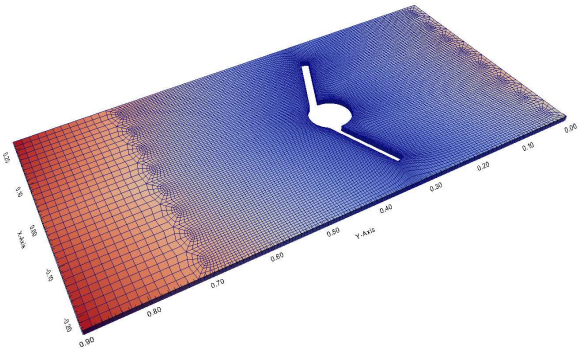
(b) Approximation of the viscosity function

Figure 6.22: Viscosity function in the experiment and its approximation in the simulation

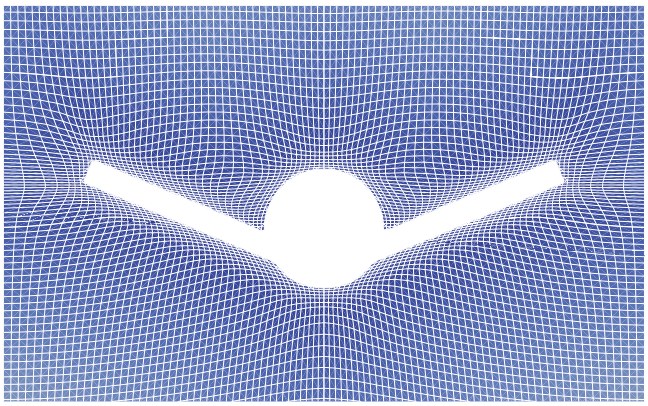
changed as well, it is identical in total length and with regard to the length and width of the shells, but the simulation uses a pseudo 2D approach in such a way that the thickness of the computational domain and the swimmer is set to a small value. Additionally, the geometry of the swimmer's head was changed from a triangular shape to a circular shape. The shells of the swimmer are modelled by thin boxes. The pseudo 2D setup of the computational domain and the swimmer geometry used in the simulation are shown in figure 6.23.

The figure shows as well how our Laplace-κ method is used to concentrate more vertices of the mesh near the surface of the swimmer geometry in order to achieve more accurate results for the hydrodynamic forces. In between time steps as in the test case of the oscillating cylinder we regularize the mesh by applying standard Laplacian smoothing and then adapting with the Laplace-κ approach again. In our simulations we computed three complete opening/closing cycles. The simulation volume was 45 mm x 90 mm x 1 mm, using 60000 time steps of $\Delta t = 0.0002s$.

In figure 6.24 we demonstrate the motion of the simmer for a full $4s$ cycle in 6 frames. A cycle consists of a slow opening half-cycle and a fast closing half-cycle which is essential for the swimmers forward propulsion [68]. We expect the swimmer to display a positive forward displacement as the configuration shown in figure 6.24 is that of a shear thickening fluid. The net displacement can be seen by comparing the swimmer position for the start frame at $0s$ and for the final frame of the cycle at $4s$. The propulsion mechanism becomes clear by examining the viscosity of the fluid for the frame at $0.3s$ of the fast closing cycle and for the frame at $2.4s$ of the slow opening cycle. In both frames the opening angle of the shells is the same, but the higher closing angular velocity produces a higher velocity gradient and hence a higher shear rate, which in turn results in a higher fluid viscosity between the shells compared to the slower open-



(a) Thin pseudo 2D domain



(b) Mesh adaptation to the swimmer

Figure 6.23: Mesh and geometry setup for the swimmer simulation

ing motion. Because of this viscosity difference the forward displacement during the closing half-cycle is higher than the backward motion during the opening half-cycle. The computed results for the displacement of the swimmer are shown in figure 6.25. As a control of our simulation and as a check for consistency we also compare experiment data for a swimmer in a Newtonian fluid with a simulation using a Newtonian fluid. For the Newtonian case a net displacement that is sufficiently close to zero is expected as the scallop theorem is in effect which should be reproduced by the numerical simulation. The displacement curves (figure 6.25) of the simulation show excellent quantitative agreement with the experimental data in both the Newtonian and shear thickening cases. The simulation results demonstrate that the net propulsion is a result of the viscosity differences during the two half-cycles, which is caused by differential apparent fluid viscosity under asymmetric shearing conditions. Furthermore, we see that our framework is able to handle cases of complex moving geometry accurately as was intended.

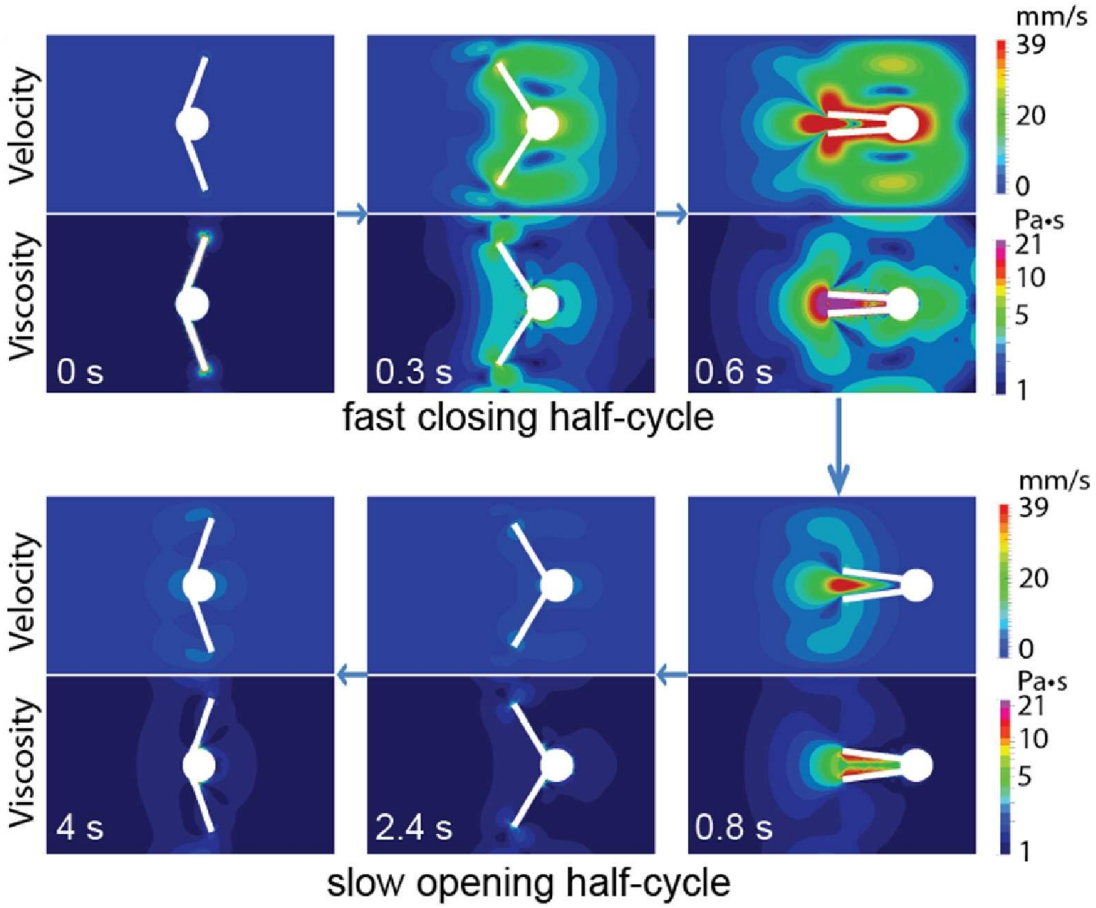


Figure 6.24: Closing and opening half-cycle of the swimmer motion with the corresponding viscosity and velocity fields.

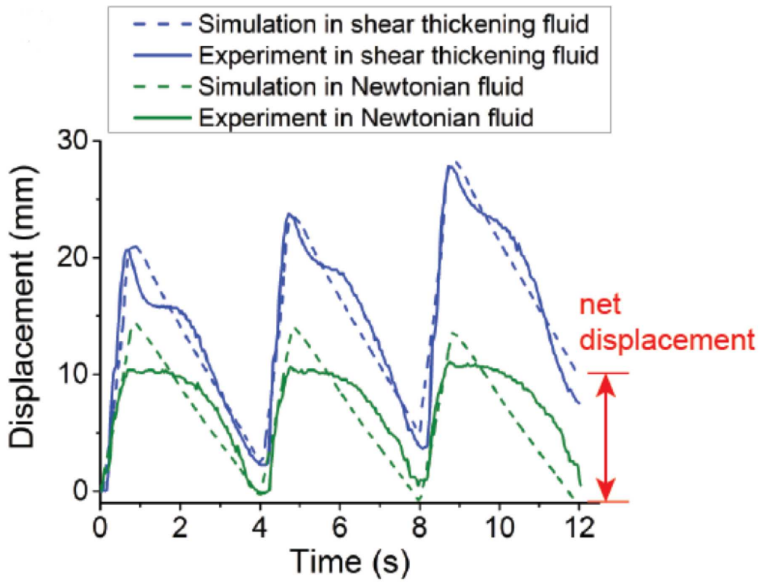


Figure 6.25: The displacement curves of the asymmetric actuated macro-scallop in shear thickening (blue) and Newtonian fluid (green) for 3 cycles. Simulation results (dashed lines) are consistent with experimental data (solid lines), where the macro-scallop exhibits net displacement in the shear thickening fluid but no net displacement in the Newtonian fluid.

6.6 Virtual Wind Tunnel

6.6.1 Test Case Description

For this test we want to demonstrate how our mesh adaptation techniques can be used to improve the resolution of geometry details in a virtual wind tunnel scenario. We insert a car geometry into a box-shaped tunnel like mesh. We create a patch around the region where the car is inserted that has a higher mesh resolution in order to reduce the total number of elements. We then use several steps of the Laplace-κ adaptation to concentrate mesh nodes near the surface of the car. An unadapted slice of the mesh is shown in figure. The number of nodes in the mesh at several refinement levels are given in table 6.5. We then compare the resolution of the details of the car for regular refinement with the geometry resolution achieved by the mesh adaptation approach. For this test case we focus more on the meshing aspect than on the flow simulation, but as a second test we show a more complex car model and compute a prototypical flow around it and show how the flow interacts with the car details by computing streamlines around the car.

Table 6.5: Number of mesh elements and vertices for virtual wind tunnel test

LEVEL	3	4
Vertices	55648	394728
Elements	43520	348160

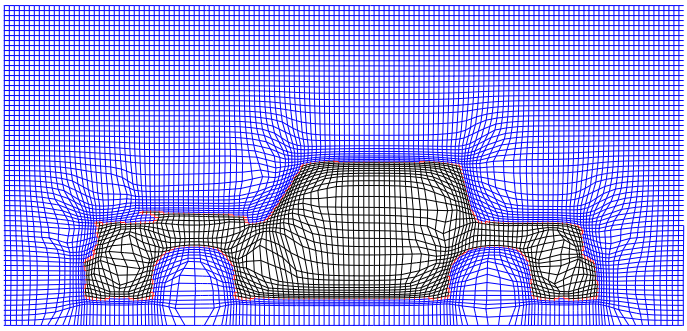


Figure 6.26: Slice of the mesh that is adapted to the simple car model

6.6.2 Simple Car Test

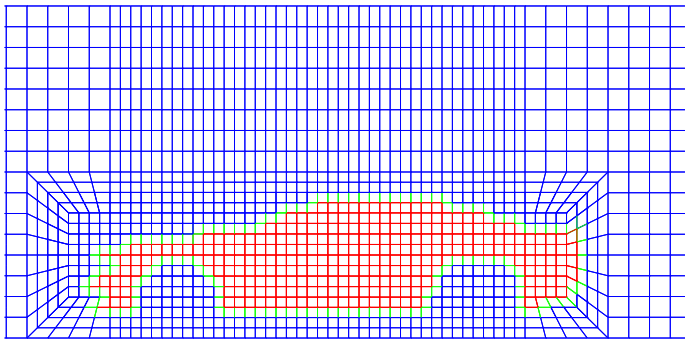
For the simple car we compare a 3D iso-volume representation of the car as 'seen' by an adapted mesh to an iso-volume representation of the car as seen by a non-adapted mesh. The iso-volume representation serves as an indicator which details of the car

Figure 6.27: Top: original car geometry (colored with distance function), middle: mesh adaptation, bottom: no adaptation

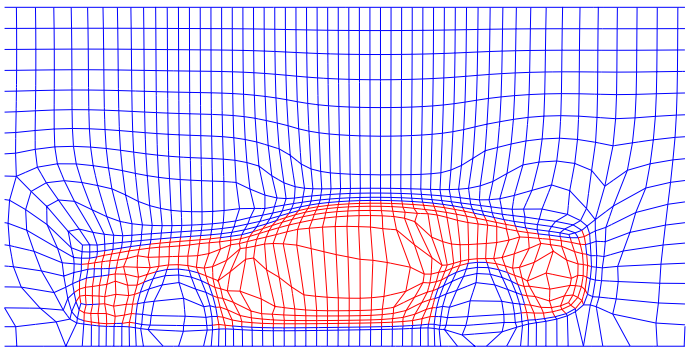
are visible to the flow solver so that a flow that takes these geometric details of the car into account can be computed. A mesh slice of the adapted mesh is shown in figure 6.26. We can already see in this mesh slice that vertices of the mesh have been relocated closer to the surface of the car and even into cavities. More consequences of mesh adaptation are revealed by the iso-surface representation in figure 6.27. These representations were built for the LVL4 mesh for both the non-adapted and adapted case. We see that in the non-adapted version the structure of the mesh is clearly visible still in the geometry representation. The windshield in the non-adapted version is not smooth at all and the steps of the hexahedral elements is visible. Cavities like in inlets for the front lights and the front grille are not captured nearly as accurately as in the adapted version. The side mirrors are not correctly seen by the non-adapted mesh, all that is present of the side mirrors are some unconnected elements. This is due to the fact that at this resolution level not enough faces of the elements are fully (meaning all vertices of the face) considered to be solid. In case of the iso-volume of the adapted mesh we can see that all major details of the car are captured and even small scale details like the inlets on the front and the side mirrors are captured using the same total number of vertices and cells. We can observe however that the features are a bit smoothed out and not as sharp as in the original. We attribute this to the fact that the adaptation approach only concentrates mesh nodes near the surface, but does not but them directly onto the surface.

6.6.3 Realistic Car Test

The car in the first test was not highly aerodynamic by current standards and had a lot of sharp edges in its geometry which makes such a type of car actually a good candidate to be represented by a regular refinement mesh. If a car is more aerodynamic with a more curved surface design the difference between a regular refinement and a mesh adaptation approach should be more visible. In figure 6.28 we see slices of the mesh with a regular regular refinement approach and with a mesh adapted version. Already in this side view of the car the mesh adapted version reveals many more geometric details of the car that are either not visible at all or only slightly visible in the regular refinement version. As before the hexahedral mesh structure is clearly visible in the inclined windshield of the car. The profile of the car in the side view is much smoother in the adapted version whereas in the non-adapted version the shape still contains sharp edges because of the underlying mesh structure. A simulation result with a streamline visualization is shown in figure 6.29. We see that the streamlines flow around the car details accurately.



(a) Base mesh without deformation



(b) Mesh adapted to the car model

Figure 6.28: Side profile of the realistic car model

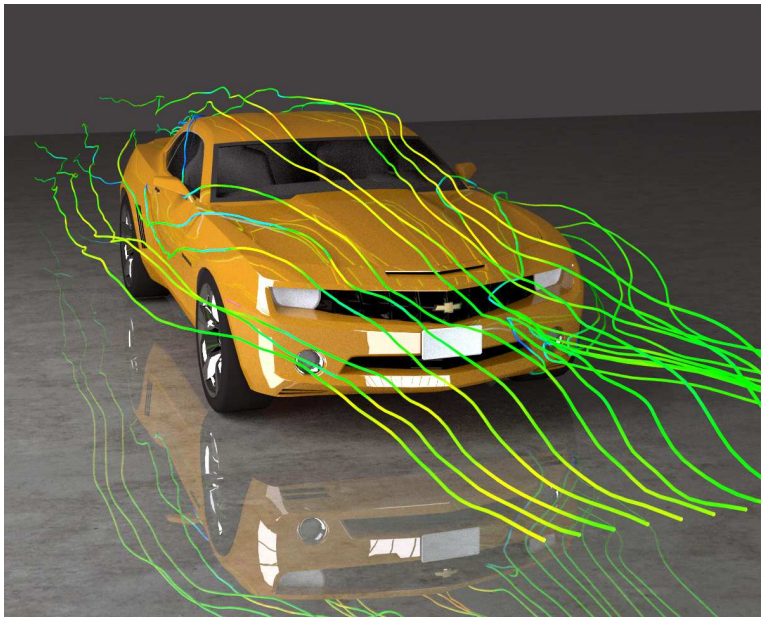


Figure 6.29: Streamline visualization of the flow around the car, streamlines interact with small scale car details

6.7 Particulate Flow Tests

6.7.1 The DGS Configuration

The next series of tests demonstrate the capabilities to handle particulate flow configurations. The first test case is a sedimentation setup that should display certain flow property of particle sedimentation. For sedimentation setups we choose the particle density to be higher than the density of the fluid. When particles sediment for example from the top of a domain to the bottom of the domain under the effect of gravity the fluid is pushed aside by the particles and fluid flow from the bottom to the top can be observed. The DGS configuration is an in-house particulate flow sedimentation computation that should reproduce this simple characteristic behavior. The effect is amplified by arranging the particles in a way that the liquid upstream caused by the sedimenting particles pass through the particle bed. An upstream flow through the particle bed would cause the particles themselves to move up again against the sedimentation direction essentially causing a twirl in the fluid and the particles. This is the flow feature that we want to observe in this configuration. The setup of the domain in dimensionless numbers is $[0, 1] \times [0, 0.25] \times [0, 1]$, the fluid density is $\rho_f = 1.0$, $\nu_f = 1.0 \times 10^{-3}$, particle density $\rho_p = 1.2$ and particle radius of $r_p = 0.0125$. The initial configuration of the simulation is shown in figure 6.32a.

DGS Results

We show the particle behavior over the course of the simulation in figure 6.32. We can see a rise in the particle bed as it gets hit by the liquid upstream. The twirl and the cause of its formation are shown in two different planes in figure 6.31. The upstream of liquid along the sides of the domain to the top is clearly visible. We have visualized the twirl in the particle bed by means of vector arrows a surface LIC which tends to make twirling motions in a velocity field more visible. Furthermore, we show the evolution of the particle z-coordinate for three particles in figure 6.30. We see that each of the three particles gets hit by liquid upstream at least once causing the z-coordinate to increase before it finally settles at the bottom.

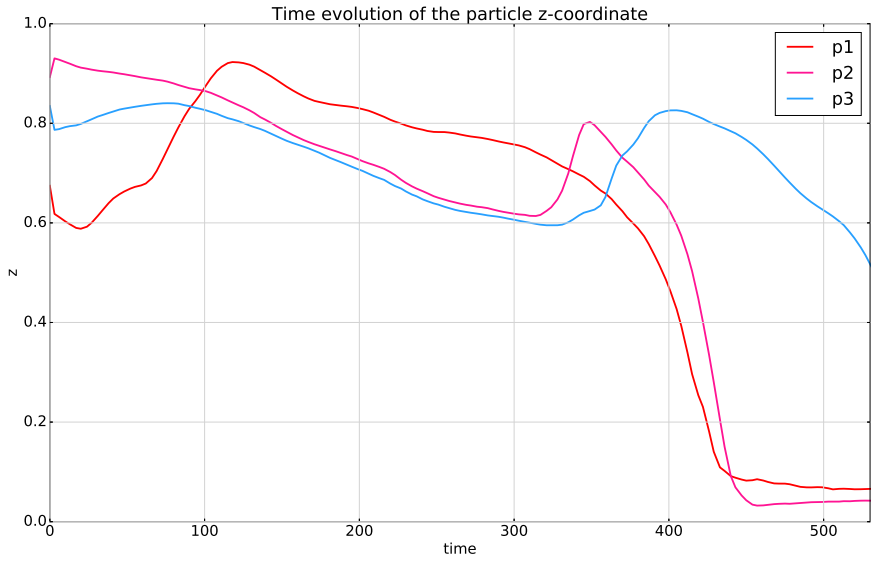
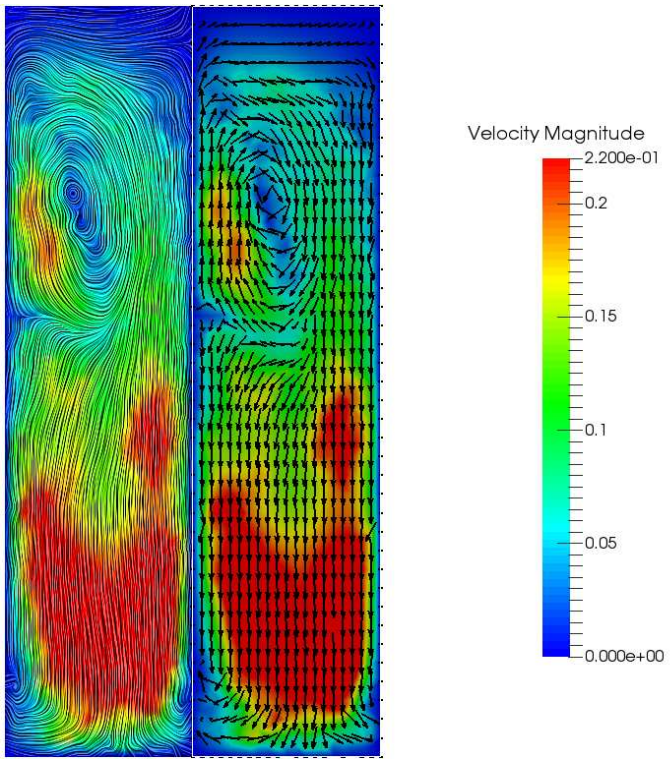
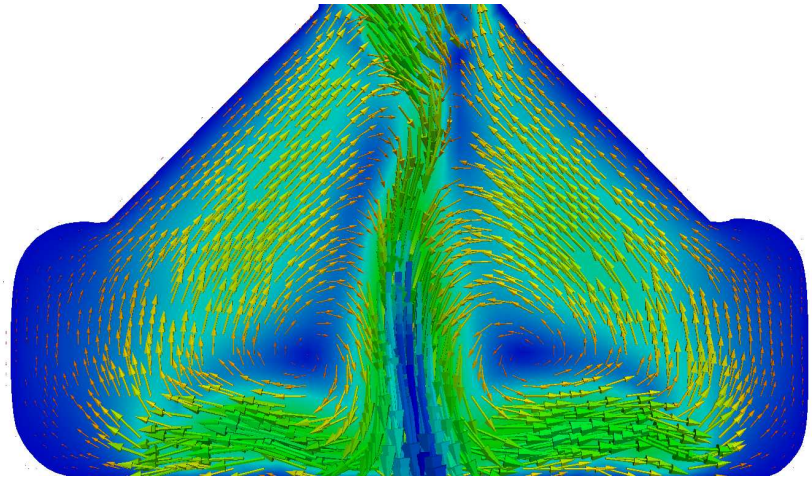


Figure 6.30: Z-coordinate evolution for three particles

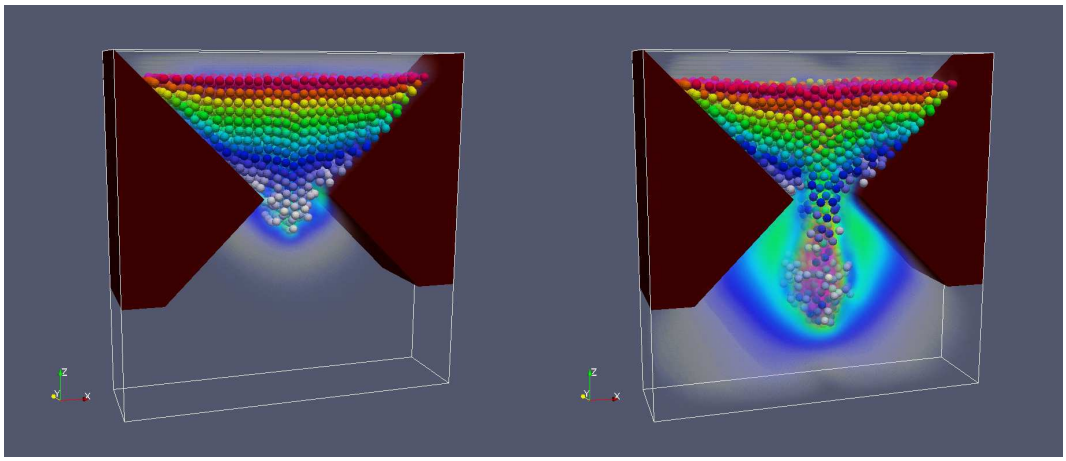


(a) YZ-Slice: Showing the twirl with surface LIC and vector visualization

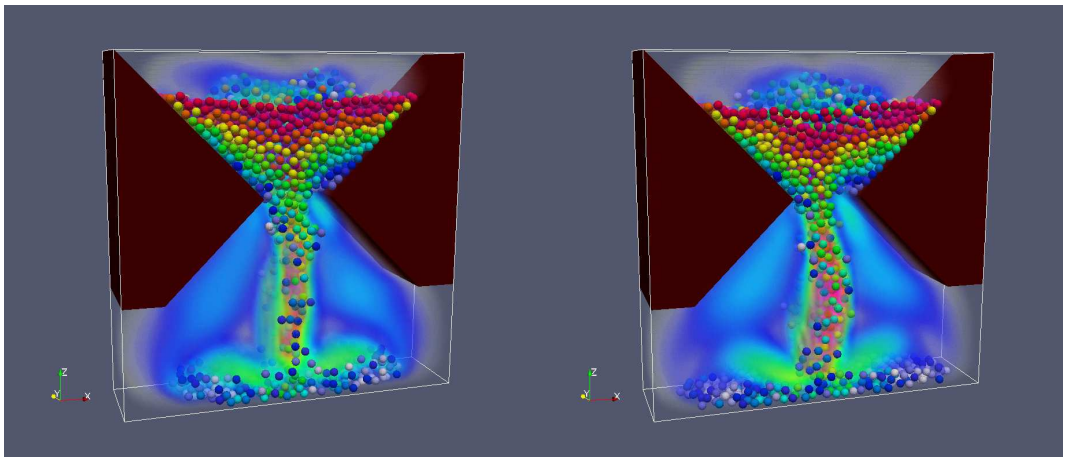


(b) XZ-Slice: Fluid upstream caused by particle sedimentation

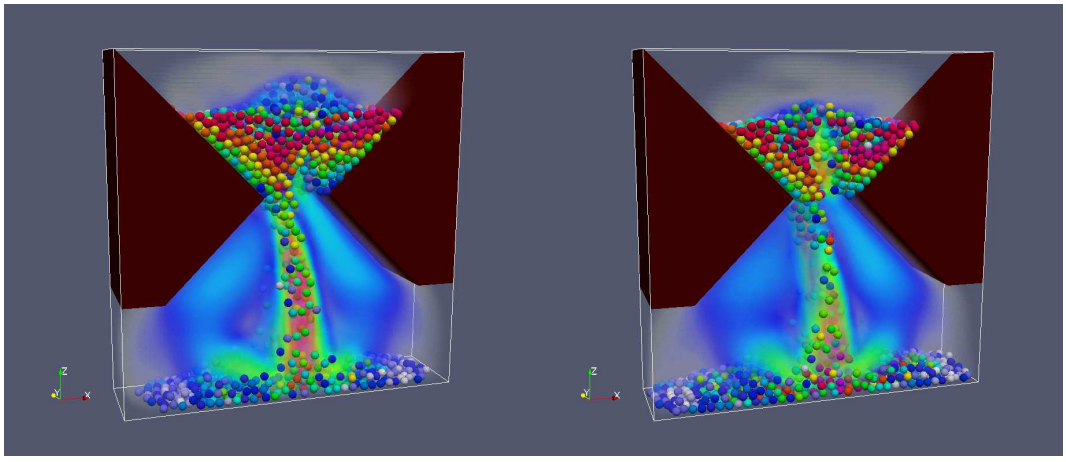
Figure 6.31: DGS: Twirl formation



(a) Initial particle distribution and beginning sedimentation



(b) Liquid upstream causes twirling in particle bed



(c) Liquid upstream continues as particles fall

Figure 6.32: DGS particle sedimentation timeline

6.7.2 Direct Numerical Simulation of a Fluidized Bed

Definition of the Test Case

This test case is based on experimental work conducted by Aguilar, Zenit and Masbernat [1]. The aim of their study was to analyze the role of particle-particle and particle-wall collisions in a fluidized bed. A fluidized bed is a system where a solid particles are immersed into a fluid in the form of a particle bed. The particles are then suspended by an upward fluid flow. In case the flow velocity exceeds a certain threshold called the mimimum fluidization velocity the particle bed is fluidized. The particle agitation is a result of the hydrodynamic forces, particle-particle and particle-wall collision forces. The setup of the experiment consists of a cylindrical glass column with a height of 60 cm and 8 cm in diameter. The fluid and particle properties are summarized in table 6.6. A fluidization velocity of $u_0 = 0.12\ m \cdot s^{-1}$ is uniformly imposed over

Table 6.6: Fluid and particle properties

Particles	Pyrex beads	$d_p = 6\ mm$	$\rho_p = 2230\ kg/m^3$
Fluid	KSCN solution	$\nu_f = 3.8 \times 10^{-3}\ Pa \cdot s$	$\rho_f = 1400\ kg/m^3$

a whole cross-section of the fluid inlet. Other important properties of the system are the theoretical terminal velocity $u_t = 0.226\ m \cdot s^{-1}$ and the particle Reynolds number $Re_p = 500$ which is defined as

$$Re_p = \frac{u_t \cdot d_p \cdot \rho_f}{\nu_f} \tag{6.5}$$

One of the quantities measured in the experiment is the evolution of the particle bed height. The height of the particle bed in turn can be used to determine the particle solid concentration. The solid concentration α_p in the fluidized bed is calculated using the following equation:

$$\alpha_p(t) = \frac{N_p 4/3\pi R_p^3}{\pi R_c^2 h_b(t)} \tag{6.6}$$

where R_c is the radius of the cylinder and N_p is the total number of particles in the fluidized bed. The solid particle concentration α_p in the fluidized bed can be used to validate the results of the simulation by comparison to the Richardson-Zaki [71] correlation. This relation describes a way to calculate the sedimentation velocity of a cloud of uniform particles in a liquid. In such a system the sedimentation velocity of a particle is influenced by the hydrodynamic forces arising from the sedimentation of the surrounding particles, the upward fluid flow and the displacement of the fluid by the particles. It was observed that the sedimentation velocity of a particle under such circumstances can be expressed as the free fall velocity of particle u_t multiplied by a correction factor that is dependent on the solid concentration α_p in the particle cloud:

$$u_0 = u_t(1 - \alpha_p)^m \tag{6.7}$$

where m is an empirical exponent called the Richardson-Zaki index that is based on the particle Reynolds number Re_p. The appropriate value for m in our case can be found in table 6.7 as determined by Richardson and Zaki. The Richardson-Zaki relation can

Table 6.7: Richardson-Zaki indices for different Re_p

Re_p	m
$Re_p \leq 0.2$	4.6
$0.2 < Re_p < 1$	$4.4Re_p^{-0.03}$
$1.0 < Re_p < 500$	$4.4Re_p^{-0.1}$
$500 \leq Re_p$	2.4

additionally be used to calculate the solid concentration α_p in a fluidized bed system by solving for α_p:

$$\alpha_{p,rz} = 1 - (\frac{u_0}{u_t})^{\frac{1}{m}}. \tag{6.8}$$

In our computations we reduced the height of the domain to 20 cm in order to save compute time, but kept the solid fraction the same which should yield same result. As a validation test the value of $\alpha_{p,rz}$ for $m = 2.4$ in our case can then be compared against the average solid concentration measured in the simulation. The averaged solid concentration in our simulation was found to be equal to 2.2 which not only compares well to the reference correlation, but also to the experiments by Aguilar [1] and the simulation of Corre et al. [11]. The evolution of the solid concentration and the bed height is shown in figure 6.33.

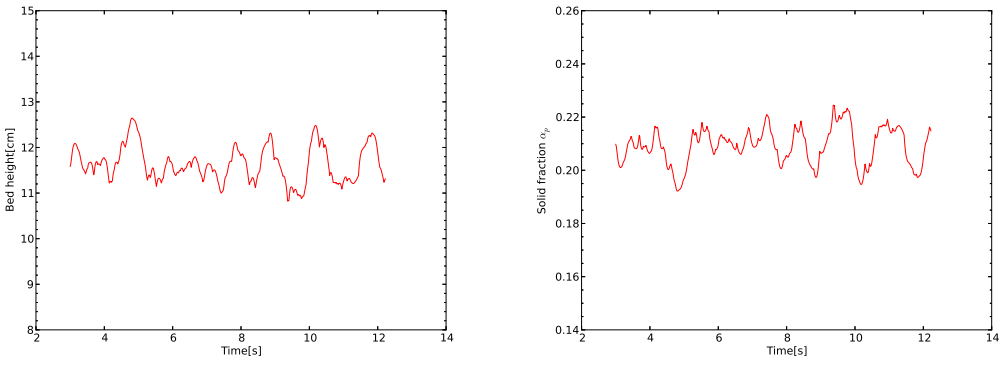
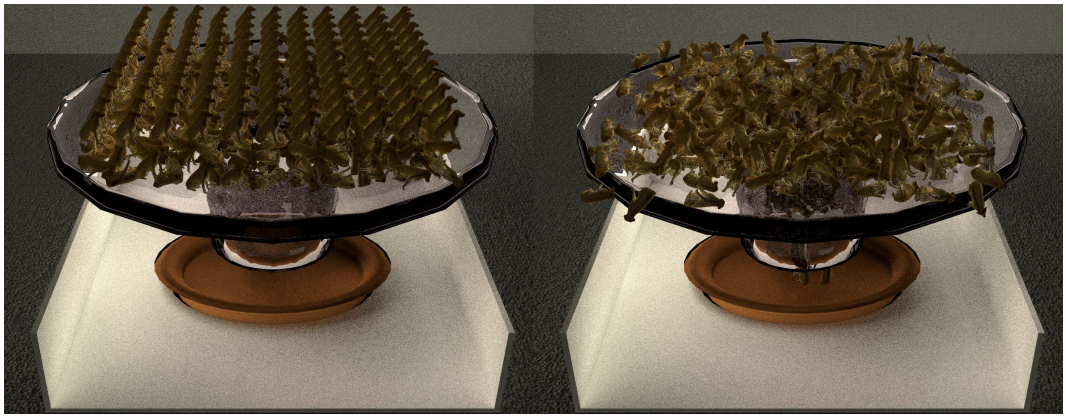


Figure 6.33: Time evolution of bed height and solid fraction α_p

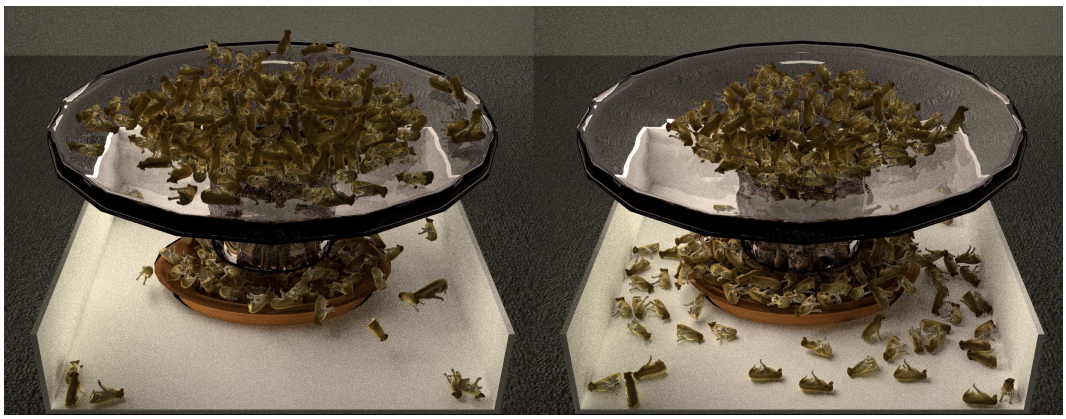
6.7.3 Complex Particles Test

This last section of our particulate flow tests we devote to the capabilities of our framework to handle particles or rigid bodies with non-spherical shapes. For the tests with

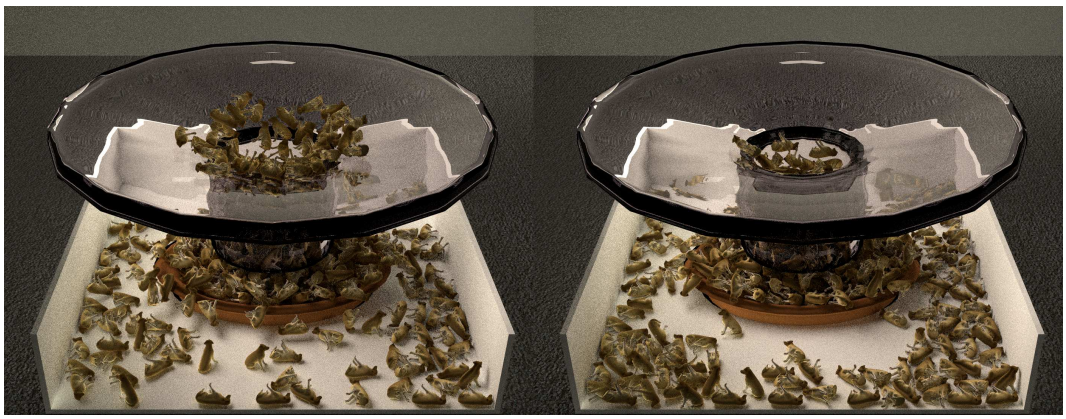
complex particles we use the distance map data structure to compute collision information. As test case for non-spherical rigid bodies we take a configuration with dog-shaped rigid bodies falling through a funnel obstacle and finally landing in a basket. The results of falling dog-shaped rigid bodies is shown in figure 6.34. Using the distance maps we see that our framework is able to detect and handle the collisions with these geometries without non-physical penetrations.
As another test computation with non-spherical geometries we consider the sedimentation of 1000 stick-like particles in a fluid, stick-like shapes are often used in fiber simulations or similar applications. In order to resolve the stick-like particles we used a mesh consisting of 4194304 elements and 4474992 vertices. In order to introduce some irregularity to the sedimentation we added a solid rigid immovable sphere in domain with that the sedimenting sticks will collide and bounce off. The result of this sedimentation simulation is shown in figures 6.35, 6.36.



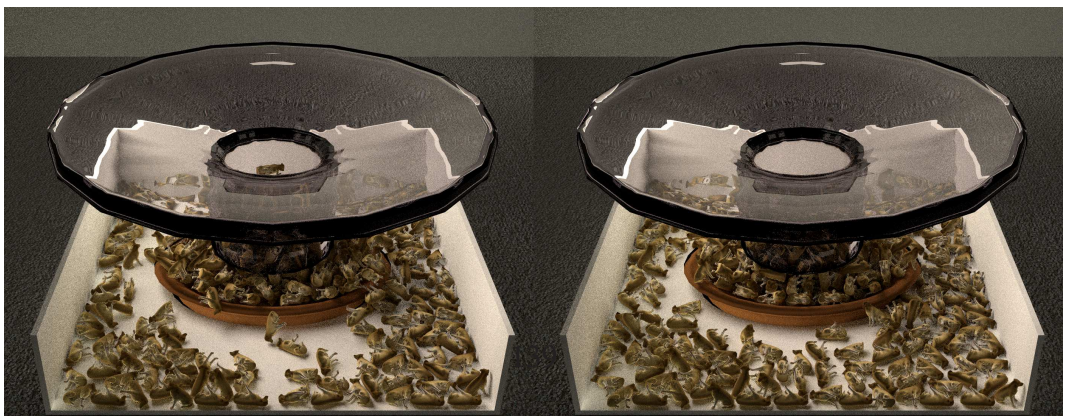
(a)



(b)

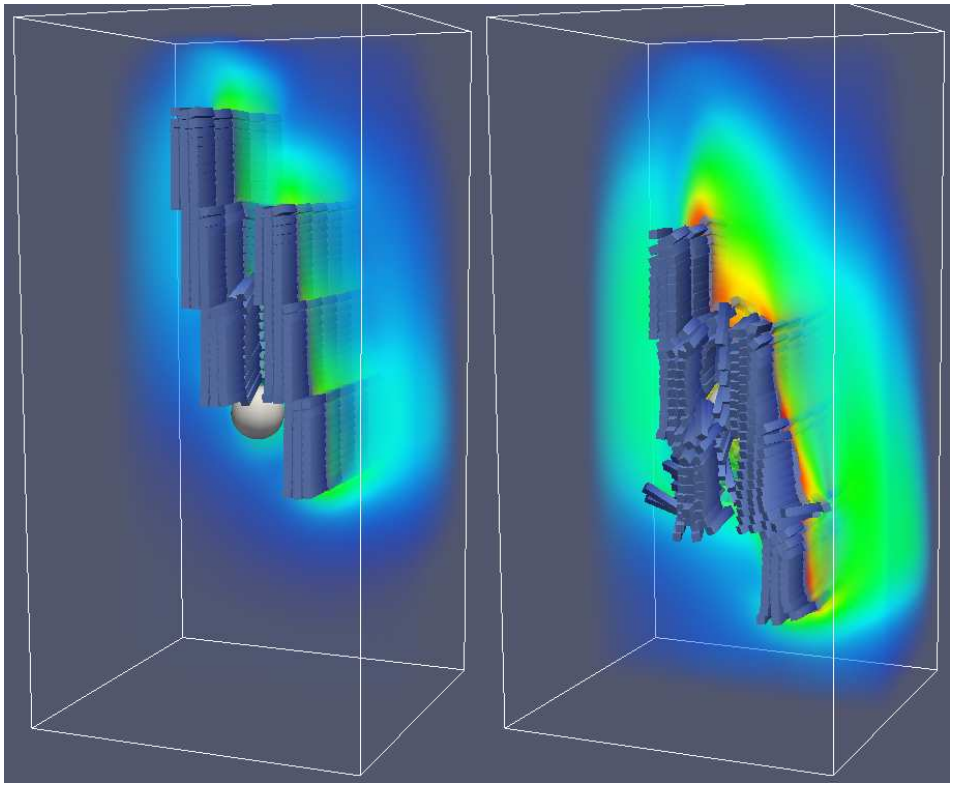


(c)

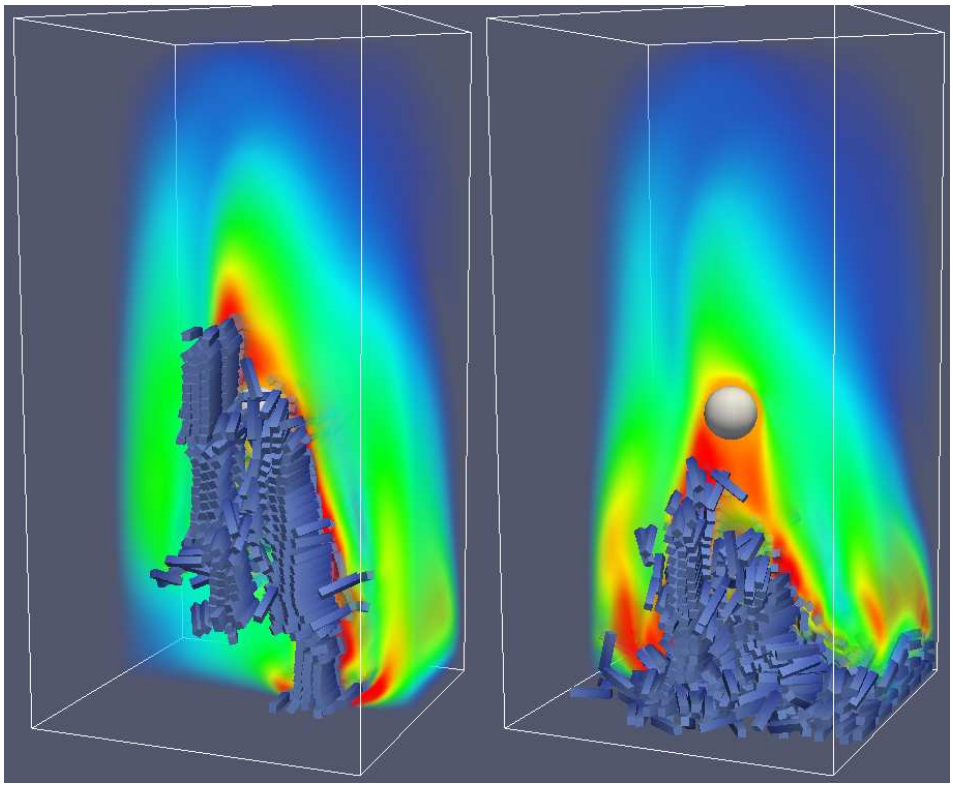


(d)

Figure 6.34: Example with complex geometries



(a)



(b)

Figure 6.35: Example with fiber-like geometries

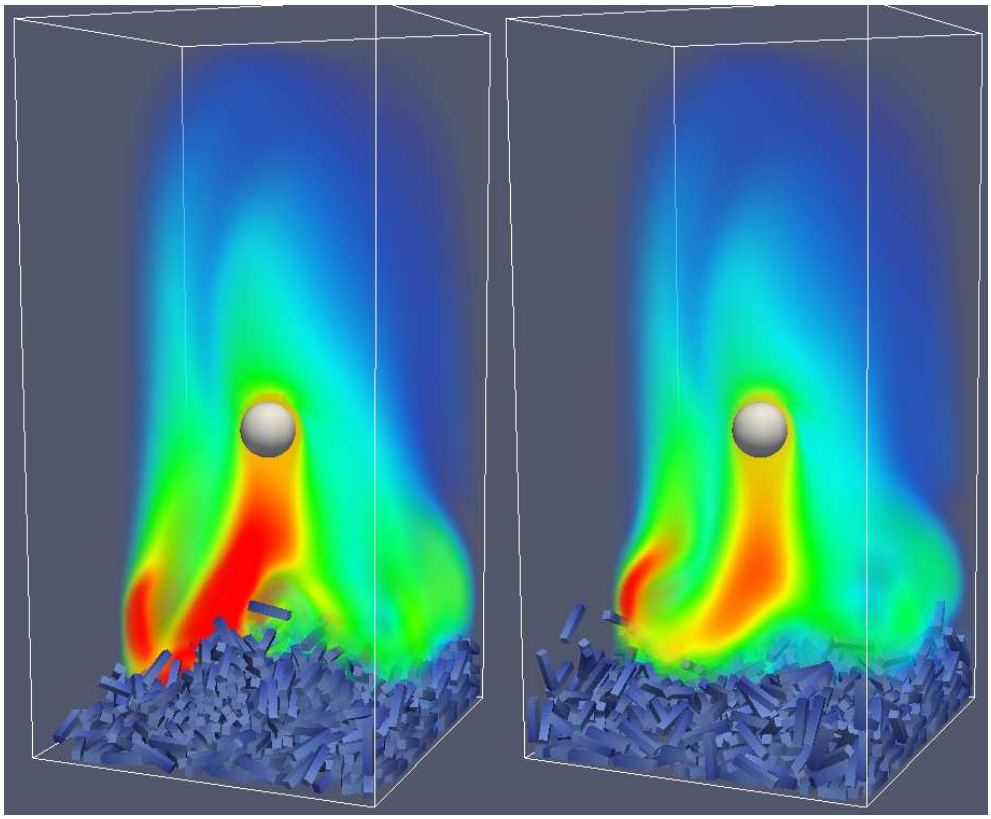


Figure 6.36: Example with fiber-like geometries

6.8 GPU Acceleration for Distance Maps and Inner Sphere Representations

In this section we want to test the most important acceleration features that we have integrated to accelerate the liquid-solid interface in the FEATFLOW solver. An integral part of calculating the hydrodynamic forces is the point containment test. We need it to determine the boundary elements in the mesh for a rigid body. The test is trivial for simple geometries, but for complex geometries the situation is more difficult, since it has to be performed for every rigid body in the simulation we have to choose efficient techniques. We will test the distance map algorithm as well as the inner sphere algorithm for meshes with different numbers of vertices. The different mesh resolutions are given in table 6.8.

Table 6.8: Number of vertices for the point containment tests

Level	1	2	3	4	5
Vertices	35937	117649	274625	912673	2146689

The number of vertices actually exceed the number of nodes that we have to deal with in real life computation. In our FEATFLOW framework we rarely have to deal with more than 100000 vertices per compute node. To see the scaling behavior of the algorithm we used higher number of nodes. For the inner sphere computations the inside of the geometry is modeled by spheres with the diameter of the cell size of the mesh. The results of the computations and a comparison between CPU and GPU versions are shown in figures 6.37, 6.38, 6.39 and 6.40.
We see that GPU versions of the algorithms significantly outperform the single CPU version of algorithms. The asymptotic scaling behavior of the algorithms is linear as expected, although for the distance map algorithm clear linear scaling sets in for higher numbers of vertices only. Although the CPU version gets outperformed by the GPU version it offers execution times that are fast enough for all realistic configurations where number of vertices per compute node is usually below 50000. If distance calculation is not a issue then the inner sphere representation leads to faster computations and similar results if the inner sphere representation is configured in such a way that the boundary spheres' diameter is about the same as the mesh cell size. If precise distance information is required the distance map data structure is preferred because with the inner sphere representation it is difficult to precisely model the exact boundary as spheres and artifacts of the spherical structure are to be expected when the distance information is used for mesh deformation. In the case of mesh adaptation the distance map offers the advantage that the distance information is more precise because it stores a distance function on a grid which can be precisely recovered using interpolation and furthermore precision can be increased by increasing the resolution of the distance map.

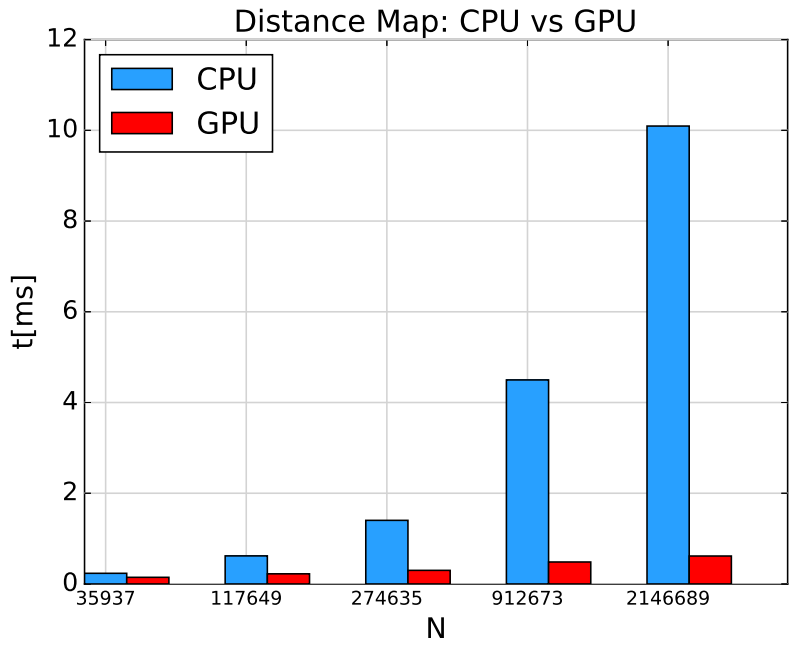


Figure 6.37: Compute time of the distance map algorithm on the CPU and GPU

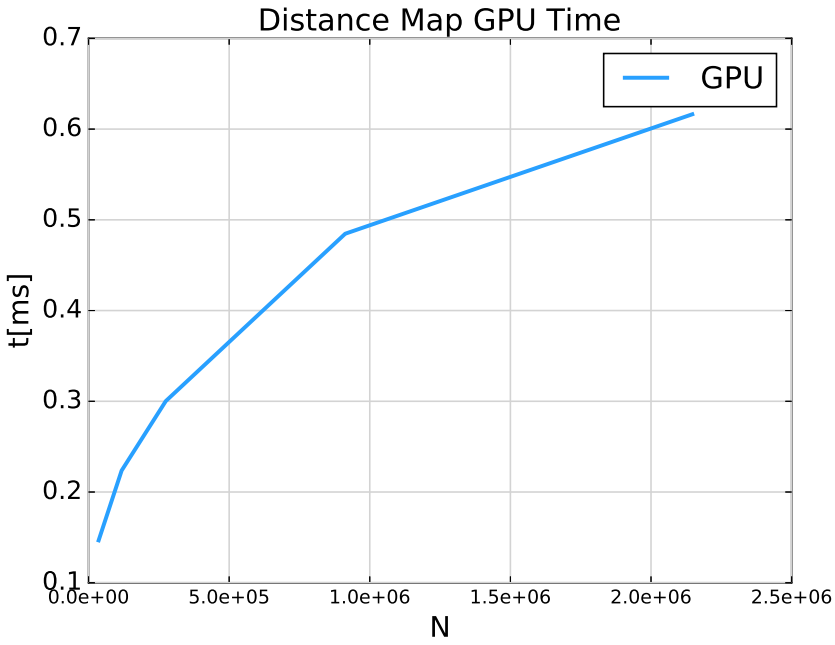


Figure 6.38: Compute time of the distance map algorithm for different mesh resolution levels

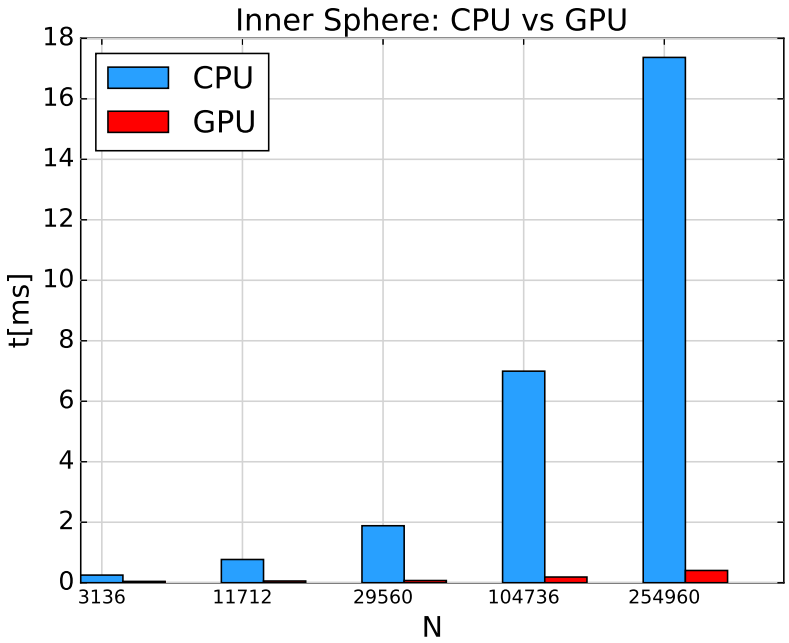


Figure 6.39: Compute time of the inner sphere algorithm on the CPU and GPU, on the x-axis we plot the number of spheres. The number of vertices of the grid is given in table 6.8

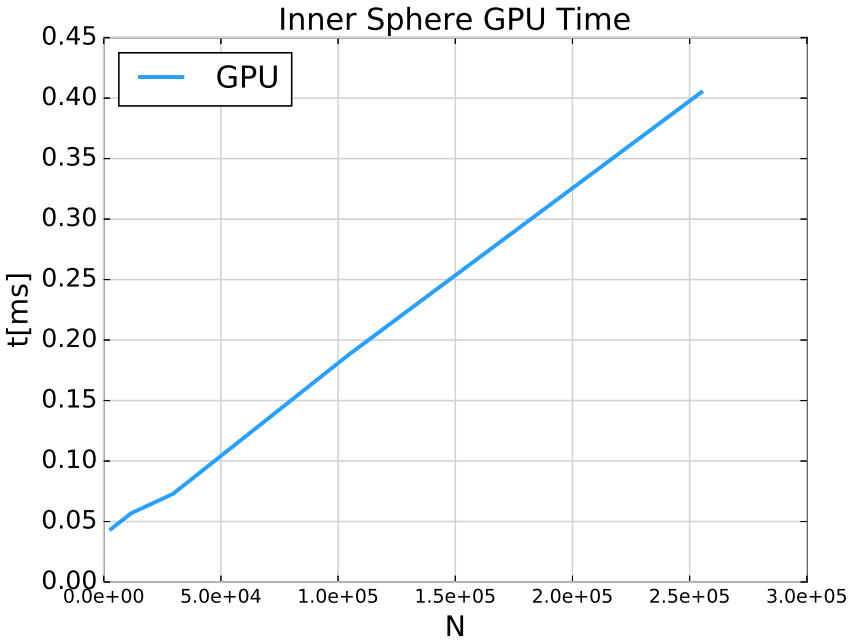


Figure 6.40: Compute time of the distance map algorithm for different number of spheres

6.9 Conclusions and Future Work

The results presented in this section show that the created simulation framework is capable of handling a wide range of particulate flow and liquid-solid simulations. The particle sedimentation tests show that results from experimental data can be reproduced accurately and also in accordance with other results of simulations from other research groups. The results show clearly the validity of the multigrid approach as the simulation results improve with increased mesh resolution. Additionally, we see how mesh adaptation can be used to increase the accuracy of results at lower refinement levels by redistribution of mesh nodes. For the sedimentation case the results with mesh adaptation at mesh refinement level l are comparable to the results of a regular refinement approach at about level $l + 2$. This is a huge improvement as the number of mesh elements on level $l + 2$ is 64 times higher and getting about the same accuracy at level l would save the compute time proportionally. In general we expect mesh adaptation approaches to increase the accuracy as much as one single step of regular refinement which still is highly beneficial. Great potential was demonstrated by the Laplace-α mesh adaptation technique which projects mesh vertices directly on the surface of an immersed geometry. This kind of adaptation produced in all respects much smoother results than a regular refinement approach and even improved on the results produced by Laplace-κ like adaptation techniques that concentrate mesh nodes near the surfaces. We can expect that if such smooth curves are used as an input to the equations of motion which are then integrated we would get highly precise results for velocity and position. The simulations of the macro-scallop swimmer demonstrate the ability of our simulation framework to handle complex moving geometries, to increase the resolution of these geometries by the mesh and again show how experimental findings can be confirmed by our simulations. The case is especially interesting as we are not only dealing with completely rigid geometry that keeps it shape, but with an immersed geometry that has its on propulsion mechanism by performing a certain motion. The result that our simulations were able to reproduce the results of the experimental swimmer allows for the justified assumption that our simulation framework would be able to handle different classes of self-propelled objects or different swimming approaches (moving flagellas, etc.) in a fully 3D simulation. The test cases of mesh adaptation to car geometries in a virtual wind tunnel setup show how the mesh adaptation approach can help to resolve small scale features of geometries. When trying to represent complex geometries the main difficulty often consists in the representation of geometry details that are relatively small compared to the total size of the immersed object. If only regular refinement is available it would be necessary to refine the mesh globally until the small scale details of the geometry can be captured by the cells of the mesh. Using mesh adaptation we can use refined mesh patches and redistribution of mesh nodes to move additional nodes into the area of interest of our simulation. We saw that just by relocating nodes we were able to capture details of the geometry with the mesh that were not represented in the computational mesh without adaptation and hence they would have no influence on the fluid. We also saw some possible areas for improve-

ment as well when we realized that the mere concentration of mesh nodes helps to capture smaller details of the geometry, but they would be 'smoothed' because we just move nodes closer to the surface of the geometry and not align faces of the mesh with the immersed geometry which would mean that the geometry would be exactly captured by the mesh including sharp edges that exist in the geometry. Our tests in the field of particulate flow show that the expected flow behavior can be reproduced by our simulations. The proposed methods for collision detection and collision force calculation could be integrated well into the simulation framework and allowed for comfortable and easy addition of complex obstacles, particles and boundaries in the simulation. The proposed methods for collision detection and collision force calculation could be integrated well into the simulation framework and allowed for comfortable and easy addition of complex obstacles, particles, rigid bodies and boundaries into the simulation. The questions how to couple the two simulation modules of CFD and rigid body simulation in the most efficient way how to setup domain decomposition and how to handle load balancing between the two simulation components definitely is a natural expansion of the aspects covered in this work and a topic of future research. Rigid bodies with complex shapes can be easily added by using distance maps or inner sphere representations. The next logical expansion in the field of complex shaped particles would be rigid particles that have mechanical joints so that parts of the geometry can move on their own or can be moved by the fluid, these new features would also be related to the simulation of objects with propulsion mechanisms that we have mentioned before. To summarize the next expansion steps to this work we mention the extension of Laplace-α mesh adaptation to general particulate flow setups which would require remeshing when the elements get too distorted as shown in section 5.7 and a projection of the old solution to the new mesh structure after remeshing. To expand on the proposed particulate flow and liquid-solid techniques we would focus on adding methods to simulate mechanical joints in our rigid bodies while keeping the assumption that the body is non-deformable. Also we consider it a natural extension to move more parts of the liquid-solid interface to the GPU as it has shown significant acceleration potential and it goes hand in hand with the acceleration of CFD simulators which are also expected to move more parts to the GPU if such hardware is available.

Bibliography

[1] AGUILAR-CORONA, A., R. ZENIT and O. MASBERNAT: *Collisions in a liquid fluidized bed*. International Journal of Multiphase Flow, 37(7):695–705, September 2011.

[2] ARDEKANI, A. M. and R. H. RANGEL: *Numerical investigation of particle-particle and particle-wall collisions in a viscous fluid*. Journal of Fluid Mechanics, 596:437–466, 1 2008.

[3] BARAFF, D.: *Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation*. SIGGRAPH Comput. Graph., 24(4):19–28, Sept. 1990.

[4] BARAFF, D.: *Coping with Friction for Non-penetrating Rigid Body Simulation*, 1991.

[5] BARAFF, D.: *Dynamic Simulation of non-penetrating Rigid Bodies*. PhD thesis, 1992.

[6] BARAFF, D.: *Fast Contact Force Computation for Nonpenetrating Rigid Bodies*. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pp. 23–34, New York, NY, USA, 1994. ACM.

[7] BARAFF, D.: *Physically based modeling: Rigid body simulation.*. Tech. Rep., Siggraph 2001 Course Notes, Pixar Animation Studios, `http://www-2.cs.cmu.edu/~baraff/sigcourse`, 2001.

[8] BICANIC, N.: *Discrete Element Methods*. John Wiley and Sons, Ltd, 2004.

[9] BLASCO, J., M. C. CALZADA and M. MARÍN: *A Fictitious Domain, Parallel Numerical Method for Rigid Particulate Flows*. J. Comput. Phys., 228(20):7596–7613, Nov. 2009.

[10] BÖNISCH, S. and V. HEUVELINE: *On the numerical simulation of the instationary free fall of a solid in a fluid.*. Computers and Fluids, 36(9):1434–1445, 2007.

[11] C. CORRE, J.L. ESTIVALEZES, S. V. and O. SIMONIN (eds.): *Direct Numerical Simulation of a Liquid-Solid Fluidized Bed*. 7th ICMF, Tampa, Florida, 2010.

[12] CAI, X.-X., B. JIANG and G. LIAO: *Adaptive grid generation based onthe least-squares finite-element method*. Computers and Mathematics with Applications, 48(7-8):1077–1085, 2004.

[13] CATE, A. TEN, C. H. NIEUWSTAD, J. J. DERKSEN and H. E. A. VAN DEN AKKER: *Particle imaging velocimetry experiments and lattice-Boltzmann simulations on a single sphere settling under gravity*. Physics of Fluids (1994-present), 14(11):4012–4025, 2002.

[14] CATTO, E.: *Iterative Dynamics with Temporal Coherence*. `http://www.bulletphysics.com/ftp/pub/test/physics/papers/IterativeDynamics.pdf`.

[15] *Coll-Det Library*. `http://cgvr.cs.uni-bremen.de/research/colldet/index.shtml`.

[16] COOK, S.: *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st ed., 2013.

[17] COQUERELLE, M. and G. H. COTTET: *A Vortex Level Set Method for the Two-way Coupling of an Incompressible Fluid with Colliding Rigid Bodies*. J. Comput. Phys., 227(21):9121–9137, Nov. 2008.

[18] COTTLE, R., J. PANG and R. STONE: *The linear complementarity problem*. Society for Industrial Mathematics, 2009.

[19] DACOROGNA, B. and J. MOSER: *On a partial differential equation involving the jacobian determinant*. Annales de l'institut Henri Poincare (C) Analyse non lineaire, 7(1):1–26, 1990.

[20] DAMANIK, H.: *Monolithic FEM techniques for viscoelastic fluids*. PhD Thesis, TU Dortmund, 2011.

[21] DRUMWRIGHT, E.: *A Fast and Stable Penalty Method for Rigid Body Simulation*. IEEE Transactions on Visualization and Computer Graphics, 14(1):231–240, Jan. 2008.

[22] EBERLY, D. H. In *Game Physics (Second Edition)*. Morgan Kaufmann, Boston, Second Edition ed., 2010.

[23] EDELSBRUNNER, H., M. J. ABLOWITZ, S. H. DAVIS, E. J. HINCH, A. ISERLES, J. OCKENDON and P. J. OLVER: *Geometry and Topology for Mesh Generation (Cambridge Monographs on Applied and Computational Mathematics)*. Cambridge University Press, New York, NY, USA, 2006.

[24] ERICSON, C. In *Real-Time Collision Detection*, The Morgan Kaufmann Series in Interactive 3D Technology. Morgan Kaufmann, San Francisco, 2005.

[25] ERLEBEN, K.: *Stable, Robust, and Versatile Multibody Dynamics Animation*. PhD thesis, University of Copenhagen, 2004.

[26] ERLEBEN, K.: *Velocity-based shock propagation for multibody dynamics animation*. ACM Trans. Graph., 26, June 2007.

[27] *FEATFLOW*. `http://www.featflow.de/`.

[28] GALDI, G. P. and V. HEUVELINE: *Lift and Sedimentation of Particles in the Flow of a Viscoelastic Liquid in a Channel*. Preprints SFB 359, Nr. 04-36, Universität Heidelberg, 2004. `http://www.iwr.uni-heidelberg.de/sfb359/PP/Preprint2004-36.pdf`.

[29] GILBERT, E., D. JOHNSON and S. KEERTHI: *A fast procedure for computing the distance between complex objects in three-dimensional space*. Robotics and Automation, IEEE Journal of, 4(2):193–203, 1988.

[30] GIRAULT, V. and P.-A. RAVIART: *Finite Element Methods for Navier-Stokes Equations: Theory and Algorithms*. Springer Publishing Company, Incorporated, 1st ed., 2011.

[31] GLOWINSKI, R., T. W. PAN, T. I. HELSA, D. D. JOSEPH and J. PÉRIAUX: *A Fictitious Domain Approach to the Direct Numerical Simulation of Incompressible Viscous Flow Past Moving Rigid Bodies: Application to Particulate Flow*. J. Comput. Phys., 169(2):363–426, May 2001.

[32] GLOWINSKI, R., T.-W. PAN, T. HESLA and D. JOSEPH: *A distributed Lagrange multiplier/fictitious domain method for particulate flows*. International Journal of Multiphase Flow, 25(5):755 – 794, 1999.

[33] GLOWINSKI, R., T.-W. PAN, V. LORENZO HECTOR JUAREZ and E. DEAN: *Numerical Methods for the Simulation of Incompressible Viscous Flow: An Introduction*. In CAPASSO, V. and J. PÉRIAUX (eds.): *Multidisciplinary Methods for Analysis Optimization and Control of Complex Systems*, vol. 6 of *Mathematics in Industry*, pp. 49–175. Springer Berlin Heidelberg, 2005.

[34] GOLDSTEIN, H., C. P. POOLE and J. L. SAFKO: *Classical Mechanics (3rd Edition)*. Addison-Wesley, 3 ed., June 2001.

[35] GRAJEWSKI, M., M. KÖSTER and S. TUREK: *Numerical analysis and implementational aspects of a new multilevel grid deformation method*. Applied Numerical Mathematics, 60(8):767–781, 2010. doi:10.1016/j.apnum.2010.03.017.

[36] GROUP, A.: *Simulation-Driven Design Benchmark Report: Getting it Right the First Time*. Tech. Rep., Aberdeen Group Inc., `http://www.reden.nl/bestanden/Aberdeen_Simulation_Driven_Design.pdf`, 2010.

[37] GUENDELMAN, E., R. BRIDSON and R. FEDKIW: *Nonconvex rigid bodies with stacking*. ACM Trans. Graph., 22:871–878, July 2003.

[38] HAHN, J. K.: *Realistic Animation of Rigid Bodies*. SIGGRAPH Comput. Graph., 22(4):299–308, June 1988.

[39] HARADA, T., M. TANAKA, S. KOSHIZUKA and Y. KAWAGUCHI: *Real-time coupling of fluids and rigid bodies*. Proc. of the APCOM, pp. 1–13, 2007.

[40] HU, H., D. JOSEPH and M. CROCHET: *Direct simulation of fluid particle motions*. Theoretical and Computational Fluid Dynamics, 3(5):285–306, 1992.

[41] HU, H. H., N. A. PATANKAR and M. Y. ZHU: *Direct Numerical Simulations of Fluid-solid Systems Using the Arbitrary Langrangian-Eulerian Technique*. J. Comput. Phys., 169(2):427–462, May 2001.

[42] JOSEPH, D. D., R. BAI, R. GLOWINSKI and V. SARIN: *Fluidization of 1204 spheres: simulation and experiments*. J. Fluid Mech, pp. 169–191, 2002.

[43] KAUFMAN, A., D. COHEN and R. YAGEL: *Volume Graphics*. Computer, 26(7):51–64, July 1993.

[44] KAČIĆ-ALESIĆ, Z., M. NORDENSTAM and D. BULLOCK: *A Practical Dynamics System*. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pp. 7–16, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[45] KINSEY, L.: *Topology of Surfaces*. Undergraduate Texts in Mathematics. Springer-Verlag, 1993.

[46] KUYPERS, F.: *Klassische Mechanik: mit über 300 Beispielen und Aufgaben mit Lösungen*. Lehrbuch Physik. John Wiley & Sons, Limited, 2008.

[47] LADD, A. J. C. and R. VERBERG: *Lattice-Boltzmann Simulations of Particle-Fluid Suspensions*, 2001.

[48] LEFEBVRE, A.: *Numerical simulation of gluey particles*. ESAIM: Mathematical Modelling and Numerical Analysis, 43:53–80, 1 2009.

[49] LLOYD, J.: *Fast Implementation of Lemke's Algorithm for Rigid Body Contact Simulation*. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 4538–4543, April 2005.

[50] MATUTTIS, H. and J. CHEN: *Understanding the Discrete Element Method: Simulation of Non-Spherical Particles for Granular and Multi-body Systems*. Wiley, 2014.

[51] MAURY: *Direct Simulations of 2D Fluid–Particle Flows in Biperiodic Domains*. J. Comput. Phys., 156:325–351, 1999.

[52] MAURY, B.: *A time-stepping scheme for inelastic collisions*. Numerische Mathematik, 102(4):649–679, 2006.

[53] MIRTICH, B.: *Hybrid Simulation: Combining Constraints and Impulses*. In *in Proceedings of First Workshop on Simulation and Interaction in Virtual Environments*. Press, 1996.

[54] MIRTICH, B.: *V-Clip: Fast and Robust Polyhedral Collision Detection*. ACM Trans. Graph., 17(3):177–208, July 1998.

[55] MIRTICH, B. and B. MIRTICH: *Rigid Body Contact: Collision Detection to Force Computation*. Tech. Rep., MITSUBISHI ELECTRIC RESEARCH LABORATORIES, 1998.

[56] MIRTICH, B. V.: *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, 1996. AAI9723116.

[57] MISHRA, B. and R. K. RAJAMANI: *The discrete element method for the simulation of ball mills*. Applied Mathematical Modelling, 16(11):598 – 604, 1992.

[58] MÜNSTER, R., O. MIERKA and S. TUREK: *Finite element-fictitious boundary methods (FEM-FBM) for 3D particulate flow*. International Journal for Numerical Methods in Fluids, 69(2):294–313, 2012.

[59] MURTY, K.: *Linear Complementarity, Linear and Non Linear Programming*. Sigma series in applied mathematics. Heldermann Verlag, 1988.

[60] NICKOLLS, J., I. BUCK, M. GARLAND and K. SKADRON: *Scalable Parallel Programming with CUDA*. Queue, 6(2):40–53, Mar. 2008.

[61] OTTMANN, T. and P. WIDMAYER: *Algorithmen und Datenstrukturen*, vol. 4. Spektrum, 2002.

[62] OUAZZI, A.: *Finite Element Simulation of Nonlinear Fluids with Application to Granular Material and Powder*. PhD Thesis, TU Dortmund, 2005.

[63] PATANKAR, N., P. SINGH, D. JOSEPH, R. GLOWINSKI and T.-W. PAN: *A new formulation of the distributed Lagrange multiplier/fictitious domain method for particulate flows*. International Journal of Multiphase Flow, 26(9):1509 – 1524, 2000.

[64] PATANKAR, N. A. and N. SHARMA: *A fast projection scheme for the direct numerical simulation of rigid particulate flows*. Communications in Numerical Methods in Engineering, 21(8):419–432, 2005.

[65] PLATT, J. C. and A. H. BARR: *Constraints Methods for Flexible Models*. SIGGRAPH Comput. Graph., 22(4):279–288, June 1988.

[66] PÖSCHEL, T. and T. SCHWAGER: *Computational granular dynamics : models and algorithms*. Springer-Verlag, 2005.

[67] PURCELL, E. M.: *Life at low Reynolds number*. American Journal of Physics, 45:3–11, 1 1977.

[68] QIU, T., T.-C. LEE, A. MARK, K. MOROZOV, R. MÜNSTER, O. MIERKA, S. TUREK, A. LESHANSKY and P. FISCHER, P. FISCHER: *Swimming by reciprocal motion at low Reynolds number*. Nature Communications, 5, 11 2014.

[69] REDON, S., A. KHEDDAR and S. COQUILLART: *Fast Continuous Collision Detection between Rigid Bodies*. Computer Graphics Forum, 21(3):279–287, 2002.

[70] REDON, S., Y. J. KIM, M. C. LIN and D. MANOCHA: *Fast Continuous Collision Detection for Articulated Models*. In *Proceedings of the Ninth ACM Symposium on Solid Modeling and Applications*, SM '04, pp. 145–156, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[71] RICHARDSON, J. F. and W. N. ZAKI: *Sedimentation and fluidisation: Part I*. Chemical Engineering Research and Design, 75, 1997.

[72] SARKAR, S., M. VAN DER HOEF and J. KUIPERS: *Fluid-particle interaction from lattice Boltzmann simulations for flow through polydisperse random arrays of spheres*. Chemical Engineering Science, 64(11):2683 – 2691, 2009.

[73] SAUER, J., D. BENZ AG, E. SCHÖMER, U. D. SAARLANDES and L. P. HOTZ: *A constraint-based approach to rigid body dynamics for virtual reality applications*. In *In Proceedings of the VRST 1998*, 1998.

[74] SCHMIDL, H.: *Optimization-based animation*. PhD thesis, 2002.

[75] SCHMIDL, H. and V. J. MILENKOVIC: *A Fast Impulsive Contact Suite for Rigid Body Simulation*. IEEE Transactions on Visualization and Computer Graphics, 10(2):189–197, Mar. 2004.

[76] SCHNEIDER, P. J. and D. EBERLY: *Geometric Tools for Computer Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.

[77] SCHWARZ, H. R.: *Numerische Mathematik*. Oxford, Clarendon Press, 1997.

[78] SHARMA, N. and N. A. PATANKAR: *A Fast Computation Technique for the Direct Numerical Simulation of Rigid Particulate Flows*. J. Comput. Phys., 205(2):439–457, May 2005.

[79] SHOEMAKE, K.: *Animating Rotation with Quaternion Curves*. SIGGRAPH Comput. Graph., 19(3):245–254, July 1985.

[80] SILCOWITZ-HANSEN, M., S. NIEBE and K. ERLEBEN: *A nonsmooth nonlinear conjugate gradient method for interactive contact force problems*. The Visual Computer, 26(6-8):893–901, 2010.

[81] STEWART, D. E.: *Rigid-Body Dynamics with Friction and Impact*. SIAM Rev., 42(1):3–39, Mar. 2000.

[82] STRANG, G.: *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 2009. Third Edition.

[83] TANG, M., D. MANOCHA, M. A. OTADUY and R. TONG: *Continuous Penalty Forces*. ACM Trans. Graph., 31(4):107:1–107:9, July 2012.

[84] TEUBER, J., R. WELLER, G. ZACHMANN and S. GUTHE: *Fast Sphere Packings with Adaptive Grids on the GPU*. In *In GI AR/VRWorkshop*, Würzburg, Germany, September 2013.

[85] TUREK, S.: *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, Berlin, 1999.

[86] TUREK, S., O. MIERKA, S. HYSING and D. KUZMIN: *Numerical Methods for Differential Equations, Optimization, and Technological Problems: Dedicated to Professor P. Neittaanmäki on His 60th Birthday*, ch. Numerical Study of a High Order 3D FEM-Level Set Approach for Immiscible Flow Simulation, pp. 65–91. Springer Netherlands, Dordrecht, 2013.

[87] TUREK, S., D. WAN and L. RIVKIND: *The Fictitious Boundary Method for the Implicit Treatment of Dirichlet Boundary Conditions with Applications to Incompressible Flow Simulations*. In *Challenges in Scientific Computing - CISC 2002*, vol. 35 of *Lecture Notes in Computational Science and Engineering*, pp. 37–68. Springer Berlin Heidelberg, 2003.

[88] WAN, D. and S. TUREK: *Direct numerical simulation of particulate flow via multigrid FEM techniques and the fictitious boundary method*. International Journal for Numerical Methods in Fluids, 51(5):531–566, 2006.

[89] WAN, D. and S. TUREK: *An Efficient multigrid-FEM Method for the Simulation of Solid-liquid Two Phase Flows*. J. Comput. Appl. Math., 203(2):561–580, June 2007.

[90] WAN, D. and S. TUREK: *Fictitious Boundary and Moving Mesh Methods for the Numerical Simulation of Rigid Particulate Flows*. J. Comput. Phys., 222(1):28–56, Mar. 2007.

[91] WAN, D., S. TUREK and L. RIVKIND: *An Efficient Multigrid FEM Solution Technique for Incompressible Flow with Moving Rigid Bodies*. In FEISTAUER, M., V. DOLEJSI, P. KNOBLOCH and K. NAJZAR (eds.): *Numerical Mathematics and Advanced Applications*, pp. 844–853. Springer, Berlin, 2003. Enumath 2003 Prague; ISBN-Nr. 3-540-21460-7.

[92] WAN, D., S. TUREK and L. RIVKIND: *An efficient multigrid FEM solution technique for incompressible flow with moving rigrid bodies*. Ergebnisberichte des Instituts für Angewandte Mathematik, Nr. 245, FB Mathematik, Universität Dortmund, Nov. 2003.

[93] WATT, A. and M. WATT: *Advanced Animation and Rendering Techniques*. ACM, New York, NY, USA, 1991.

[94] WELLER, R.: *New Geometric Data Structures for Collision Detection*. Dissertation, University of Bremen, Germany, october 2012.

[95] WELLER, R. and G. ZACHMANN: *Inner Sphere Trees for Proximity and Penetration Queries*. In *2009 Robotics: Science and Systems Conference (RSS)*, Seattle, WA, USA, June 2009.

[96] WELLER, R. and G. ZACHMANN: *ProtoSphere: A GPU-Assisted Prototype-Guided Sphere Packing Algorithm for Arbitrary Objects*. In *ACM SIGGRAPH ASIA 2010 Sketches*, pp. 8:1–8:2, New York, NY, USA, Dec. 2010. ACM.

[97] WELLER, R. and G. ZACHMANN: *Inner Sphere Trees and Their Application to Collision Detection*. In COQUILLART, S., G. BRUNNETT and G. WELCH (eds.): *Virtual Realities*, ch. 10, pp. 181–202. Springer (Dagstuhl), 2011.

[98] WITKIN, A., K. FLEISCHER and A. BARR: *Energy Constraints on Parameterized Models*. SIGGRAPH Comput. Graph., 21(4):225–232, Aug. 1987.

[99] YAMANE, K. and Y. NAKAMURA: *Stable penalty-based model of frictional contacts*. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 1904–1909, May 2006.