

# **A Partitioned Approach for Fluid-Structure Interaction on Cartesian Grids**

Bernhard Gatzhammer



Technische Universität München

Computational Science and Engineering  
(Int. Master's Program)

Master's Thesis

*A Partitioned Approach for Fluid-Structure  
Interaction on Cartesian Grids*

Bernhard Gatzhammer

1st examiner: Univ.-Prof. Dr. Hans-Joachim Bungartz  
2nd examiner: Univ.-Prof. Dr. Ernst Rank  
Assistant advisor: Dr. Miriam Mehl  
Thesis handed in on: 29<sup>th</sup> of August 2008

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

München, 29.08.2008

---

Bernhard Gatzhammer



# Abstract

This work is about the computer-based simulation of problems from fluid-structure interaction (FSI). Fluid-structure interactions are a multiphysics problem and combine fluids, structures, and their bi-directional interactions. Instead of solving one specific problem or applying one specialized approach to the field of FSI, the thesis rather strives to provide a software environment supporting any kind of investigations into and solutions of partitioned FSI problems. It does so by providing a coupling environment for two independent simulation programs, one specialized on the simulation of fluids and the other one on structures. The special idea behind this tool is to encapsulate all coupling functionality and tools into a unit called coupling supervisor, which minimizes the amount of work necessary to prepare a simulation program for coupled simulations. This is achieved with a client-server based software architecture, hiding the coupled simulation programs from each other by a redirection of all communication to the coupling supervisor and the use of a dedicated coupling mesh for the communication of coupling data. In order to approve the maturity of the coupling tool, it must be validated by the simulation of a well defined test scenario defining quantities for comparison with approved results. This leads to the second part of this work, the preparation of a fluid simulation program for the simulation of an FSI benchmarking scenario. To qualify a fluid solver for partitioned FSI simulations, it must be able to handle dynamic obstacles and to transfer data from its discretization mesh to that of the coupling supervisor. Finally, numerical results are presented, showing the influence of implemented features and giving some first ideas about the expected results of the FSI benchmark scenarios.

# Zusammenfassung

Diese Arbeit handelt von der rechnergestützten Simulation von Problemen der Fluid-Struktur-Wechselwirkung (FSW). Fluid-Struktur-Wechselwirkungen gehören zur Gruppe der Mehrfeldprobleme und vereinen Fluide, Strukturen und deren bi-direktionale Wechselwirkungen in einem gemeinsamen Rahmen. Anstatt ein bestimmtes Problem zu lösen oder einem bestimmten Ansatz zur Lösung von FSW Probleme nachzugehen, versucht diese Arbeit, eine Software Umgebung zur Verfügung zu stellen, die beliebige Nachforschungen in und Lösungsansätze für das Feld der Fluid-Struktur-Wechselwirkungen unterstützt. Dies wird ermöglicht, indem eine Umgebung zur Kopplung von zwei unabhängigen Simulationsprogrammen zur Verfügung gestellt wird, wobei eines der Programme auf die Lösung von Problemen der Fluidmechanik und das andere auf die Lösung von Problemen der Strukturmechanik spezialisiert sein muss. Die besondere Idee hinter dem Ansatz dieser Arbeit ist es, die komplette Kopplungsfunktionalität und alle unterstützenden Werkzeuge in einer extra Einheit, genannt Coupling Supervisor, zu kapseln und so den Aufwand, der benötigt wird um ein Simulationsprogramm auf gekoppelte Simulationen vorzubereiten, zu minimieren. Dies wird mit einer Client-Server basierten Software Architektur erreicht, die die Simulationsprogramme voneinander verbirgt und jegliche Kommunikation auf den Coupling Supervisor umleitet. Zusätzlich wird ein speziell zum Austausch von Kopplungsdaten bestimmtes Gitter eingeführt, über das sämtliche Kopplungsdaten kommuniziert werden. Um die Ausgereiftheit des Coupling Supervisor zu überprüfen, muss ein ausreichend beschriebenes FSW TestszENARIO simuliert werden, das die Validierung der Ergebnisse mit allgemein anerkannten Ergebnissen ermöglicht. Dies führt direkt zum zweiten Teil dieser Arbeit, der Vorbereitung eines Strömungslösers auf die Berechnung von FSW Benchmark Szenarien. Damit ein Strömungslöser für partitionierte FSW Probleme geeignet ist, muss er dynamische Hindernisse verwalten und Daten zwischen seinem Diskretisierungsgitter und dem des Coupling Supervisor abbilden können. Zum Schluss der Arbeit werden einige numerische Ergebnisse präsentiert, die die implementierten Funktionalitäten aufzeigen und eine erste Idee über die zu erwartenden Ergebnisse des FSW Benchmarks liefern.

# Acknowledgement

This work would not have been possible without the help of many people and I want to express my gratitude to them at this place.

I thank Miriam Mehl for being the supervisor of my thesis. I always felt welcome by her to ask any questions or mention concerns on my mind and was often surprised by her commitment to help me. Her thorough proofreading of my thesis was a great support for me.

I'm thankful to Markus Brenk for introducing me to the software tools and methodologies used in this thesis. His support enabled me to have a good starting point for the tasks of this thesis.

Two people, which I bombarded with questions, were Tobias Neckel and Ioan Lucian Muntean. I thank both for their invaluable help. Without them, this thesis would not have found any good end.

Another person I also want to mention here is Stefan Kollmannsberger from the chair of Computation in Engineering. Although I was in contact with him only for a short time during this thesis, I appreciated his help very much.

Finally, I want to thank all people from the chair of Informatik V for creating such a supporting and friendly atmosphere and my examiners Prof. Hans-Joachim Bungartz and Prof. Ernst Rank for their efforts to examine this work.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	About Fluid-Structure Interactions . . . . .	1
1.2	Simulation Approaches on Fluid-Structure Interaction . . . . .	2
1.3	Research Background of this Thesis . . . . .	3
1.4	Chapter Overview . . . . .	4
<b>2</b>	<b>FSI*ce - a Fluid-Structure Interaction Coupling Environment</b>	<b>5</b>
2.1	Basic Ideas behind FSI*ce . . . . .	5
2.1.1	Requirements on FSI*ce . . . . .	5
2.1.2	Client-Server Concept . . . . .	6
2.1.3	Coupling in Space . . . . .	7
2.1.4	Communication Layers . . . . .	9
2.2	Restructuring FSI*ce . . . . .	11
2.2.1	The GNU Build System Manages the Build . . . . .	12
2.2.2	FSI*ce Project Frame Before Restructuring . . . . .	12
2.2.3	Motivation for a New Structure . . . . .	13
2.2.4	FSI*ce Project Frame After Restructuring . . . . .	15
2.3	Coupling Schemes in Time - Design of a Coupling Unit . . . . .	17
2.3.1	Ways to Couple . . . . .	17
2.3.2	Software Design and Functionality of the New Coupling Unit . . . . .	18
2.4	Modifications on the Application Programming Interface . . . . .	22
<b>3</b>	<b>About Fluids and Structures</b>	<b>26</b>
3.1	Physical Model for Fluids . . . . .	26
3.1.1	The Continuum Principle . . . . .	27
3.1.2	Lagrangian and Eulerian View . . . . .	28
3.1.3	Reynolds Transport Theorem . . . . .	30
3.1.4	Conservation of Mass . . . . .	31
3.1.5	Conservation of Momentum . . . . .	31
3.1.6	The Navier-Stokes Equations for Incompressible Flows . . . . .	33
3.1.7	Rendering the Navier-Stokes Equations Dimensionless . . . . .	34
3.2	Physical Model for Structures . . . . .	36
3.2.1	Strain Tensor . . . . .	36
3.2.2	Stress Tensor . . . . .	38
3.2.3	Kinetics and the Relation between Strain and Stress . . . . .	39

3.3	The Wet Surface - Coupling Equations . . . . .	40
<b>4</b>	<b>Simulation Tools and Employed Methods</b>	<b>42</b>
4.1	F3F for Flow Simulations . . . . .	42
4.1.1	Cartesian Grids . . . . .	43
4.1.2	Spatial Discretization by the Finite Volume Method . . . . .	44
4.1.3	Time Discretization by the Chorin Projection . . . . .	47
4.1.4	Computation of Forces on Obstacles . . . . .	49
4.2	AdhoC for Structure Simulations . . . . .	50
4.2.1	The Finite Element Method for Structure Computations . . . . .	50
4.2.2	High Order Shape Functions . . . . .	52
<b>5</b>	<b>F3F - Preparing a Fluid Solver for FSI Simulations</b>	<b>54</b>
5.1	Extensions of Boundary Types . . . . .	54
5.1.1	Fundamental Boundary Types . . . . .	54
5.1.2	A Quasi-2D Scenario in Three Dimensions . . . . .	56
5.2	Re-Implementation of FSI Functionality . . . . .	58
5.2.1	Motivation for the Re-Implementation . . . . .	58
5.2.2	Re-Implemented Functionality . . . . .	60
5.2.3	Handling of Multiple Obstacles . . . . .	60
5.3	Further Work Done . . . . .	61
5.3.1	Internal FSI and Implicit Coupling . . . . .	63
5.3.2	Exchanging the Convective and Diffusive Operators . . . . .	63
5.3.3	Integrating the New Coupling API . . . . .	64
<b>6</b>	<b>Numerical Results</b>	<b>65</b>
6.1	Internal FSI Simulations . . . . .	65
6.2	Outflow and Slipwall Boundaries . . . . .	65
6.3	FSI Benchmark . . . . .	67
6.3.1	Benchmark Description . . . . .	67
6.3.2	First Results . . . . .	71
<b>7</b>	<b>Conclusions</b>	<b>73</b>
7.1	What has been Achieved . . . . .	73
7.2	Outlook on further Development . . . . .	74
<b>A</b>	<b>Explicit and Implicit methods for the numerical solution of ODEs</b>	<b>76</b>
<b>B</b>	<b>Introduction to Tensor Notation</b>	<b>80</b>
<b>C</b>	<b>F3F Code Details</b>	<b>81</b>

# 1 Introduction

## 1.1 About Fluid-Structure Interactions

Fluid-structure interaction is a problem from multiphysics, where two, in classical theory separated fields interact with each other in a bi-directional sense, namely fluids and structures. Despite their common origin of continuum mechanics, fluids and structures both have their own background of theories and mathematical descriptions, for example by partial differential equations. Brought on a computer, again specialized strategies for discretization and numerical solutions are applied for each field. In the past, each sub-problem offered already enough complexity and computational challenges to completely bind the focus of research. Increasing computing power and highly developed numerical algorithms now enable considering fluids, structures and their interaction coupled together in one scenario.

The application possibilities for fluid-structure interaction computations are indeed endless. In mechanical engineering one strives to optimize the drag on moving bodies like cars or aircrafts. Building engineers want to find out how bridges or sky scrapers react on certain wind patterns or optimize tent-like structures for lightweight roofs<sup>1</sup>. A physician might be interested in the blood flow through flexible arteries to investigate the source of heart attacks. Many more examples can be found, and just the plain possibility to perform bi-directionally coupled simulations of fluids and structures will let arise applications in fields formerly not taken into account.

All the mentioned problems exhibit a high complexity and in most cases no analytical solution can be found. Experiments are possible for some cases, but they are expensive with regards to time and money and the output of information provides only limited means for design optimization and scientific understanding. Thus, numerical simulations can provide another helpful approach to get further insights into these kinds of problems and to find optimized solutions.

---

<sup>1</sup>A famous example of such a structure can be found in the Olympia stadium of Munich.

## 1.2 Simulation Approaches on Fluid-Structure Interaction

When trying to solve a coupled problem on a computer by a simulation, one first has to find out the approach best suited for the given problem. The approaches to the field of coupled problems can be subdivided into two classes, one being called *monolithic* and its counterpart, *partitioned* (cf. Figure 1.1). The first approach treats the coupled problem as a whole, sets up one system of equations describing both fields including interactions and, hence, steps ahead simultaneously in time with each field. Its advantages are possibilities for error control, an inherent coupling mechanism and robustness as consequence. However, the different characteristics of the individual fields cannot be taken into account. The latter approach strives to solve each physical field separately, with communication of interface data in-between to connect the components. For this approach, a variety of strategies exists on how to couple the interacting simulation programs. On the one hand, this gives greater flexibility on how to solve the problem. On the other hand, it introduces a factor of complexity and uncertainty with regards to optimal coupling strategies. While a monolithic approach might be perfectly suited for a software house developing a package offering FSI capabilities, in research work a higher level of flexibility is needed, which is in favor of the partitioned approach. This thesis focusses on a partitioned approach for fluid-structure interaction, leaving the monolithic approach out of any further discussion.

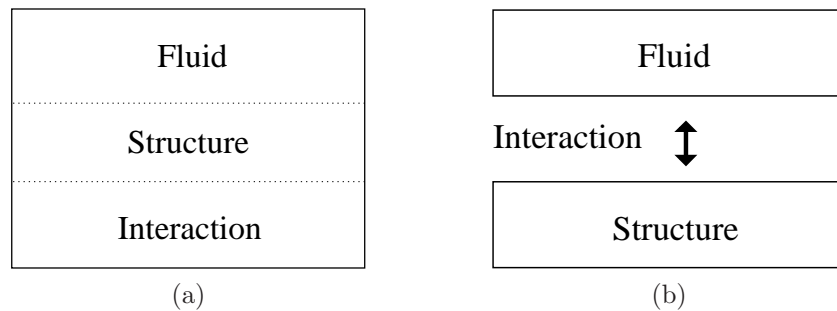


Figure 1.1: Schematic view of a monolithic (a) and a partitioned (b) approach.

Let us first look a bit closer at the advantages and difficulties of partitioned approaches, in order to get motivated to dive into the details of it. One key argument for partitioned approaches is their flexibility. Flexibility not only in terms of the choice of coupling algorithms as mentioned before, but also in the choice of software for solving the structure and fluid equations. Imagine the possibility to simply take two solvers and let them work together as if they were always meant to do so. Of course, this plug-and-play philosophy requires some coupling environment that simplifies this procedure and some small intrusions into the solver's codes will be inevitable, too. The coupling procedure itself is not by any means a straightforward thing. Since the solvers are separated, they both use their own discretization method, leading to nonmatching grids, different

stability requirements on time step widths and requirements of interface data. A structure solver will most probably use finite elements on unstructured grids as discretization scheme, while a fluid solver might use a finite volume approach on Cartesian grids. The need for interpolation methods arises, in order to map data of one mesh to the other. In addition, a fluid with low viscosity might be highly turbulent and require very small time steps to stay stable, while the structure could advance with much larger steps. A kind of subcycling might improve the situation by allowing the fluid solver to accumulate several small time steps on its own. Finally, the fluid solver might be trimmed to compute forces on obstacles, but the structure solver requires stresses as interface values. A transformation is necessary, which should fulfill certain properties such as conservation of total force or a certain level of accuracy of the computed interface values. We easily recognize, that the benefits gained by the partitioned approach introduce a high level of complexity that must be understood and controlled in order to get reasonable results. This makes it a challenging task to provide a satisfying coupling tool for partitioned fluid-structure interaction.

### 1.3 Research Background of this Thesis

This work is basically a continuation of the dissertation of Markus Brenk [4]. He started his thesis at the Universität Stuttgart in 2002 and finished it after a move to the TU München in 2007. His doctorate position was sponsored by the group 493 of the Deutsche Forschungsgemeinschaft (DFG), “Fluid-Struktur-Wechselwirkung: Modellierung, Simulation, Optimierung”<sup>2</sup>, where he resided in the project “Numerische Simulation von Fluid-Struktur-Wechselwirkungen auf kartesischen Gittern”<sup>3</sup>.

The projects in the DFG research group 493 are given in the following list, where project 8 and 10 have been in close cooperation with our project.

- P1: Mehrgitterverfahren zur effizienten numerischen Simulation von Fluid-Struktur-Wechselwirkungen, TU Darmstadt.
- P2: Adaptive FE-Methoden zur Fluid-Struktur-Kopplung, Universität Heidelberg.
- P3: Monolithische ALE-Verfahren und gekoppelte Mehrgitterlöser für Fluid-Struktur-Wechselwirkung, Universität Dortmund.
- P4: Numerische und experimentelle Untersuchung der Fluid-Struktur-Wechselwirkung für turbulente dreidimensionale Strömungen - Prinzipkonfiguration FLUSTRUC, Universität Erlangen-Nürnberg.
- P6: Numerische Simulation von Fluid-Struktur-Wechselwirkungen auf kartesischen Gittern, TU München.

---

<sup>2</sup>English: Fluid-Structure Interaction: Modelling, Simulation, Optimization

<sup>3</sup>English: Numerical Simulation of Fluid-Structure Interaction on Cartesian grids

- P8: Lattice-Boltzmann Ansätze zur Simulation bidirektionaler Fluid-Struktur-Probleme mit Turbulenz, TU Braunschweig.
- P9: Effiziente Ansätze bei Fluid-Struktur-Wechselwirkungen mit großen Deformationen und Topologieänderungen, TU München.
- P10: Elemente hoher Ordnung in der Fluid-Struktur-Wechselwirkung, TU München.
- P11: Sensitivitätsanalyse und Optimierung im Kontext partitionierter Verfahren der Fluid-Struktur-Wechselwirkung, TU München.

### 1.4 Chapter Overview

In Chapter 2, a tool to couple a fluid simulation program with a structure simulation program is introduced. After describing the main features of it, the restructuring work and extensions done within this thesis are described. To prepare the mathematical understanding of simulation tools used to simulate fluid-structure interaction problems, Chapter 3 gives the physical background of fluids, structures and the coupling conditions. In Chapter 4, the fluid simulation program F3F and the structure simulation tool AdhoC are introduced, with F3F in the main focus of descriptions. Chapter 5 describes the work done to F3F during this thesis, reaching from extensions of boundary types to restructuring work on the obstacle handling functionalities. Numerical results are presented in Chapter 6, which also describes the setup of the FSI Benchmarks. Finally, in Chapter 7, the work of this thesis is summarized and a short outlook is given on future work.

# 2 FSI<sup>ice</sup> - a Fluid-Structure Interaction Coupling Environment

FSI<sup>ice</sup> (pronounced F-S-Ice) is a software tool that strives to supply an environment for coupling together a fluid simulation program with a structure simulation program and let them simulate a problem from fluid-structure interaction. Hence, its name is “Fluid-Structure Interaction Coupling Environment” (FSI<sup>ice</sup>). In the sense of a partitioned simulation approach (cf. Section 1.2), its goal is to provide a set of communication means and coupling methodologies to simplify the coupling of the two simulation programs.

This chapter starts with a description of the basic ideas behind FSI<sup>ice</sup> that have been investigated by Markus Brenk in [4]. The second part of this chapter deals with the source code project structure of FSI<sup>ice</sup>, that had to encounter a major restructuring during the work on this thesis. In the third part of this chapter, we will look at different coupling schemes in time for partitioned simulations and see how the new coupling unit of FSI<sup>ice</sup> implements a versatile coupling strategy. Finally, the last part of this chapter is about the modifications on the application programming interface for users of FSI<sup>ice</sup>, discussing the importance and flexibility of a such an extended interface.

## 2.1 Basic Ideas behind FSI<sup>ice</sup>

This section describes the main ideas and requirements that motivated the development of FSI<sup>ice</sup>, and the software design concepts used to implement it. We will see, that a so-called “plug-and-play” philosophy is one essential feature of FSI<sup>ice</sup>, which is supported by a client-server based communication architecture and a dedicated coupling mesh for surface data exchange. The material of this section is mainly inspired from the work of Markus Brenk [4], in which most parts mentioned here are described in more detail.

### 2.1.1 Requirements on FSI<sup>ice</sup>

When Markus Brenk thought about the requirements put on FSI<sup>ice</sup> to make it a valuable coupling tool, he investigated an extensive list of features that should be supported. A summary of what he found is given here, with putting special emphasize on the re-

quirements most important for this thesis.

**User Requirements** The user of FSI\*ce must be able to modify his solvers'<sup>1</sup> codes in a simple, well-defined way with only few efforts, while gaining a maximum amount of coupling functionality like coupling schemes or mesh interpolation methods. A more advanced user must have the possibility to utilize special features of his simulation programs, such as high order mesh elements, by overwriting standard functionalities of FSI\*ce.

Because FSI\*ce is a service tool coupling together two foreign simulation programs, special emphasize must be put on portability of FSI\*ce onto different machines and support for different coding languages. Only then, it can become a widespread tool for many users. The goal is to provide a superior build system (cf. Section 2.2.1) and to support integration into C, C++ and Fortran codes, which the most codes are written in.

Common features of simulation programs such as parallel execution or checkpointing must be supported by FSI\*ce, while the possibility to perform a coupled simulation on distributed computing resources, such as grid services, provides additional value for the user.

**Software Development Requirements** Since FSI\*ce is a piece of software, it of course has requirements from the developer side, who must maintain, change and extend it. A modular and layered design layout with possibilities to perform certain diagnoses is hence a must.

Despite FSI\*ce aims at coupling together a fluid and a structure solver, a software design supporting arbitrary types of solvers is highly attractive for extending the application of FSI\*ce to further multiphysics simulation fields in the future.

### 2.1.2 Client-Server Concept

The user requirements, listed in the Section 2.1.1 motivate a software architecture fully encapsulating the coupling functionalities in an extra unit separated from the solvers. This unit, henceforth called *coupling supervisor*, acts as the only direct coupling partner for the participating solvers and enables thus a “plug-and-play” mechanism. This mechanism requires a solver to be modified only once for coupled simulations, and afterwards being able to work together with any other solver, hidden behind the coupling

---

<sup>1</sup>“Solver” is used here as just a shorter form for simulation program. (This is in contrast to other contexts, where it refers to the unit in a simulation program or other calculation tool, that solves a given system of equations.)

supervisor.

The software architecture supporting these features in the best way is a client-server based one, as illustrated in Figure 2.1. The coupling supervisor acts as client, sending simulation requests to the solvers acting as servers. The servers have to fulfill their given tasks and need to send some information about their current status to the coupling supervisor, enabling it to optimally control the coupled simulation. The coupling supervisor chooses the appropriate time step length, triggers output and decides upon the kind of coupling step (cf. Section 2.3) to be performed next by a solver. A maximal amount of flexibility, in terms of coupling control is achieved by choosing this communication layout. New coupling methods can be integrated into the coupling supervisor without the necessity of adapting the solvers' codes.

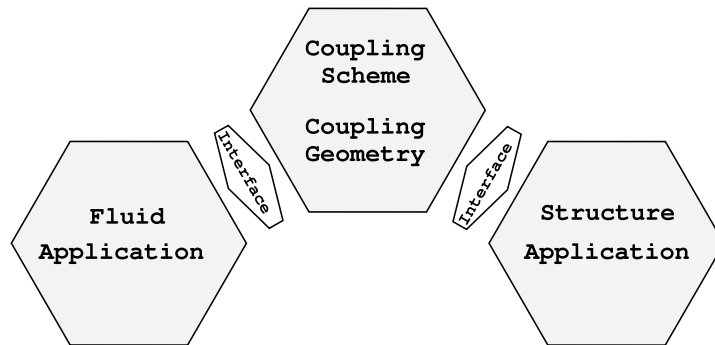


Figure 2.1: Client-server concept of FSI\*ce. A central control unit called coupling supervisor acts as client in the coupled simulation and is sending requests to the simulation applications of fluid and structure domain. These applications act as servers fulfilling the request of the client and sending the computed data back to it. All communication taking place between the client and the servers, is defined by a unique interface.

### 2.1.3 Coupling in Space

**A coupling mesh** One essential property of partitioned simulation approaches to FSI problems is, that both, the fluid and the structure solver, use their own individual spatial discretization meshes. These meshes will be non-matching at the coupling interface in general, as Figure 2.2 illustrates, and will require projection and interpolation methods to transfer information in-between the meshes. It is one major task of FSI\*ce to supply a mechanism supporting this exchange of coupling interface data from mesh to mesh, without introducing a direct dependency of the solvers.

Facilitated by the client-server architecture described in the previous Section 2.1.2, FSI\*ce introduces an additional mesh for interface data exchange, called *coupling mesh*. Every simulation program participating in the coupled simulation, as well as the coupling supervisor itself, have to hold an instance of this coupling mesh. A simulation

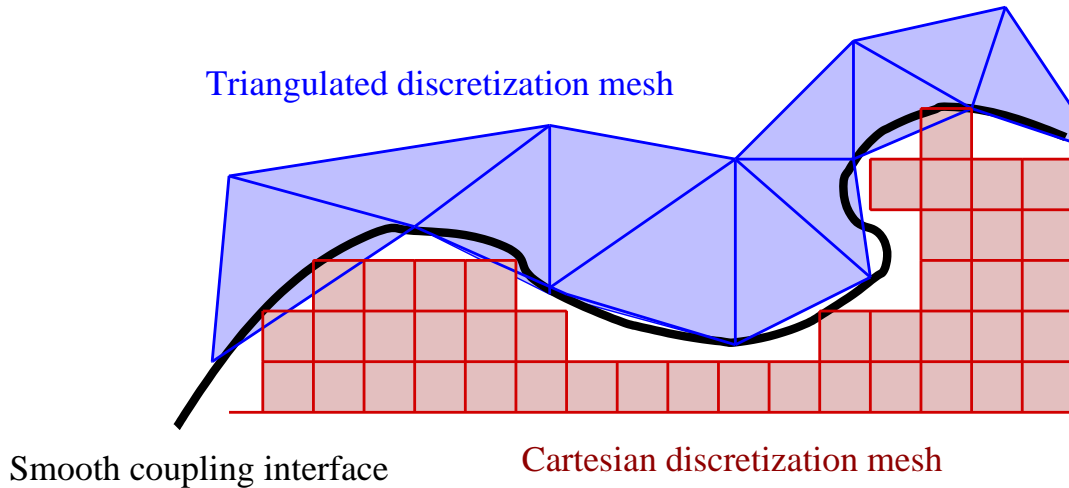


Figure 2.2: Example for two non-matching meshes at the coupling interface. The real interface geometry is smooth, but can only be represented by the discretized meshes. These are partially overlapping or contain gaps at the coupling interface. Interpolation and projection methods are necessary to transfer data from one mesh to the other.

program has to transfer its coupling interface data to and from this coupling mesh, in order to communicate information to the coupling supervisor, which in turn transfers this information to the other participating simulation program. By this indirect communication mechanism of mesh interface data, the solvers are kept independent from each other.

The coupling mesh exists in two implementation variants: `FSI_Mesh` and `GeoBase_Mesh`. `FSI_Mesh`, the newer version, is derived from `GeoBase_mesh`, which is taken from the simulation program `AdhoC` (cf. Chapter 4). Both meshes are based on triangulated vef-graphs, consisting of vertices, connected by edes, forming triangular faces. The following description is restricted to the mesh `FSI_Mesh`, which should be considered as the first choice of coupling mesh to be used. `FSI_Mesh` can store, besides the topological information of the mesh itself, datasets of simulation data to be exchanged. A dataset assigns an additional value to each vertex of the `FSI_Mesh`, that can consist of either one or three numbers, representing scalar or vectorial values. Each dataset also has a defined data type, that can be either `FSI_Attribute`, for integer values, or `FSI_Value`, for floating point values. The functionality of the coupling mesh is encapsulated in the library `FSImesh` a well-defined interface, described in the work of Markus Brenk [4].

**Geometry creation in Cartesian grids and closest neighbor search** `FSI*ce` provides tools for converting the geometry of the coupling mesh into a representation on a Cartesian grid, for supporting grid transformations of a solvers grid and for finding nearest neighbors of a solver’s mesh vertices to the vertices of the coupling mesh. To perform these operations efficiently, an octree is employed (cf. remark 2.1). The octree holds

information about the geometry represented by the coupling mesh. Every cell of the octree has an attribute indicating whether it lies fully within the fluid or solid domain or on the surface. In the case of Cartesian solver discretization meshes, this information can be queried and directly used to convert the geometry of the coupling mesh, based on triangles, into a Cartesian grid conform geometry. Furthermore, each octree cell relates all triangles of the coupling mesh, that are fully, or partially contained within it. Given a point within the simulation domain, the octree functionality is able to find the nearest point on a triangle of the coupling mesh, projected orthogonally using barycentric coordinates, in an efficient way by exploiting inheritance within the octree.

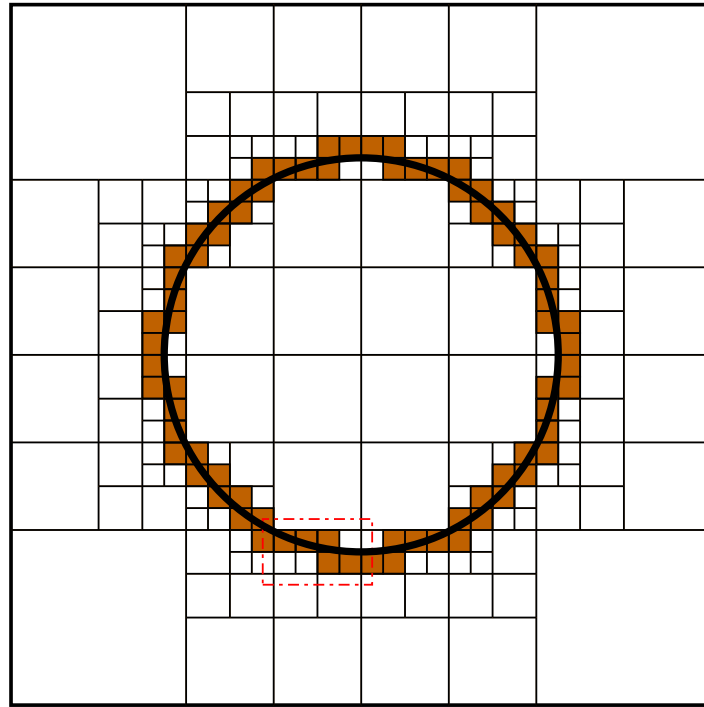
**Remark 2.1.** Spacetrees and efficient geometry processing

A spacetree is a hierarchical treelike data structure for representing geometries on Cartesian grids. The three dimensional variant of a spacetree, originating from the subdivision of a three dimensional cell into eight equal subcells, is called octree. Figure 2.3 shows an example of a two dimensional geometry discretized by an adaptive Cartesian grid, and its corresponding representation by a quadtree, the two dimensional analogon of an octree (with four subcells per refinement). Only the cells containing a part of the geometry's surface are refined recursively up to a certain level. This enables to store a highly accurate representation of a smooth geometry on Cartesian grids, without the need to refine the whole grid, and reduces the growth of storage necessary to  $O\left(\frac{1}{h^2}\right)$  for the octree, and to  $O\left(\frac{1}{h}\right)$  for the quadtree, when  $h$  is the minimal meshwidth.

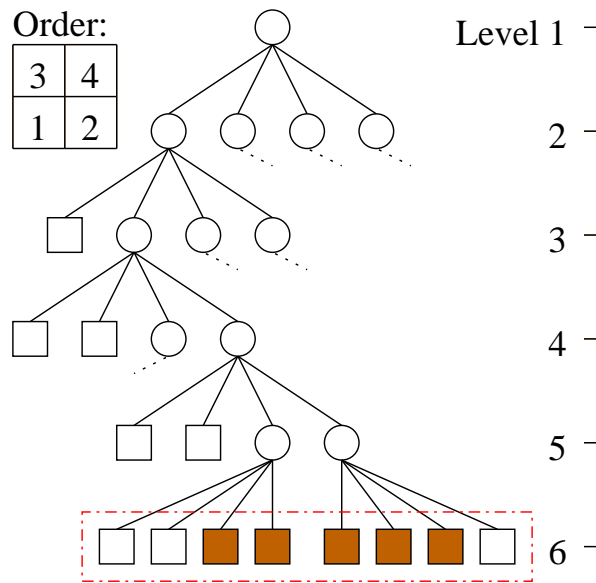
In addition, a spacetree implicitly contains information about neighboring cells in its tree-structure. This can be used to find neighboring objects assigned to the spacetree cells.

## 2.1.4 Communication Layers

As described in the software development requirements in Section 2.1.1, a layered communication design, enabling the separation of different communication concepts, is another important part of the design of FSI\*ce. Figure 2.4 shows the software design of the communication libraries of FSI\*ce. The most basic layer for communication is of course the network layer, which enables communication at all. On top of it an implementation of socket- or MPI- (Message Passing Interface) based communication is used, that sends and receives messages from the network. The MshPI (Mesh Passing Interface) layer provides functionality to send and receive coupling specific data, like the coupling mesh or basic datatypes. For the solvers coupled via FSI\*ce, an additional layer, called FSIcom, is added. It provides a defined interface to be integrated into the solver's codes.



(a)



(b)

Figure 2.3: Adaptively refined Cartesian grid (a) with extract of the corresponding quadtree representation (b). Filled cells in brown represent the circular geometry on the Cartesian grid. The ordering of the quadtree nodes to the Cartesian grid cells is indicated on the top left of (b). A refined cell is represented as node (circle) in the spacetree and a not further refined cell as a leaf (square). The red dashdotted rectangle indicates corresponding grid cells and quadtree nodes. The level of grid refinement is indicated by a ruler on the right side of (b), with level 1 being the outer frame of the domain in (a).

We will see more about this user interface in chapter 2.4. The separation of concepts into layers enables the developer to change the implementation of one of the layers, without the need to adapt the source code in the other layers, which are connected via an FSI<sup>ce</sup> internal interface. For example, the use of a third communication mechanism for communicating in grid environments could be added to the MshPI layer, without needing to adapt the FSIcom layer.

While the two bottom layers, network and transport, are just used as external functionality by FSI<sup>ce</sup>, the MshPI and FSIcom layer are implemented as modules of FSI<sup>ce</sup>. To work on this module structure was part of my work and will be discussed in more detail in the next section.

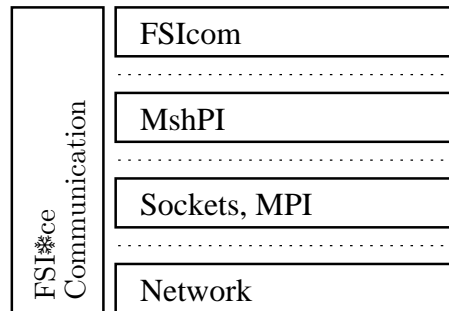


Figure 2.4: Layered communication software design of FSI<sup>ce</sup>.

## 2.2 Restructuring FSI<sup>ce</sup>

At the beginning of this thesis, the FSI<sup>ce</sup> program code was given in the form of separated program libraries. Among the functionalities implemented, there were some belonging together but being actually separated into different libraries, while others seemed to be not optimally placed in a specific library. A single source code project frame containing all parts of FSI<sup>ce</sup> in a modified form, seemed to be more advantageous than a distributed one, and hence, the source code packages of FSI<sup>ce</sup> were restructured within this thesis. In this section, we look at the source code project as it has been originally, the motivation to restructure it, and finally, the structure FSI<sup>ce</sup> obtained after the restructuring. Both, the old and the new project frame, are enabled by a superior build system, the GNU<sup>2</sup> build system, which I introduce briefly beforehand, to give the necessary background for the following chapters.

---

<sup>2</sup>GNU stands for “GNU is not Linux”, which is a recursive acronym.

### 2.2.1 The GNU Build System Manages the Build

FSI\*ce uses the GNU Build System [16] in order to create libraries and executables out of source code. This set of tools is the standard build system for free software development under Linux. Who ever had to install an open source package on a Linux system, has encountered the typical command sequence

```
./configure
make
make install
```

These commands are part of the GNU Build System visible to the user of a software. They are created by the same build system and execute further scripts of it to accomplish the build. The first command `configure`, among many other things, adjusts the build configuration to the target hardware, performs tests for certain libraries necessary during the installation and can take optional flags to enable or disable certain features of the program to be compiled. The second command, `make`, compiles the code with chosen configurations and `make install` copies the created libraries and executables to the installation directories.

In contrast to a user, the software developer has to start on a lower level in the GNU build system, by writing the proper configuration script that is later processed when typing `configure`. In addition, he needs to create descriptions that give information about the libraries and executables being created during the build, in files with name `Makefile.am`. These files are first converted to input Makefiles for the configuration script, and only at the moment of the installation into hardware conform standard Makefiles processed when typing `make` and `make install`.

Of course, to make the build system work requires some initial effort, but the benefits gained from it are definitely worth it in the long run, especially for projects aiming to run on different hardwares and offering its users an easy installation process with configuration possibilities. In particular the ability to switch on and off the compilation of parts of a program is used in when restructuring FSI\*ce.

### 2.2.2 FSI\*ce Project Frame Before Restructuring

At the beginning of this thesis, FSI\*ce was composed of separated source code packages, as illustrated in Figure 2.5.

The two left packages, `FSImesh` and `FSIcom`, are libraries mainly written by Markus Brenk [4] in cooperation with Professor Rank et. Al. As described in Section 2.1.3 and 2.1.4, `FSImesh` provides the basic functionality for exchanging the coupling mesh and basic data types. The library `FSIcom` provides the an application programming interface

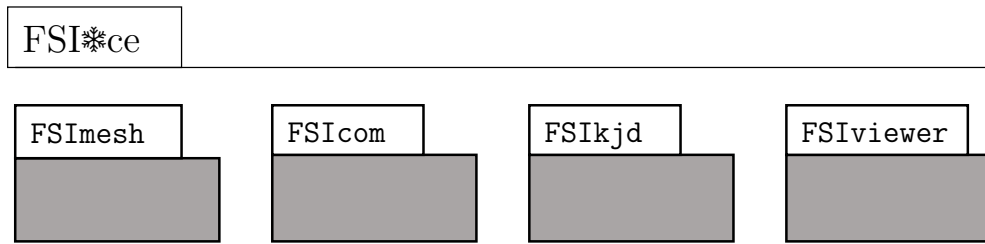


Figure 2.5: FSI\*ce source code packages before restructuring. Each package has its individual source code project, responsible for creating a part of FSI\*ce. The user of FSI\*ce needs to gather all packages in order to create the coupling supervisor and libraries.

(API) for a user of FSI\*ce (cf. Section 2.4) and the actual coupling supervisor program, acting as the client in the client-server based software architecture.

FSIkjd and FSIviewer are written by Klaus Daubner, who did his master thesis under supervision of Markus Brenk [9]. He introduced an octree (cf. Section 2.1.3) to find for any given point the nearest neighboring point on a triangle of the coupling mesh and projecting its data to this point in a consistent manner. This functionality is included in the library FSIkjd. In addition, he introduced a GUI (Graphical User Interface) viewer which is able to visualize data from the fluid solver and the structural solver during an FSI simulation run. While both, the octree functionality and GUI are very helpful, he introduced also other features contradicting the philosophies of FSI\*ce, which are closely bound to the GUI viewer.

The big advantage of having different independent source code modules separated into several packages is, that once a package is completed, it can be installed and must not be touched any more. This prevents accidental changes and rises the barrier for non-justified hacks to speed up some lazy development.

### 2.2.3 Motivation for a New Structure

The new source code structure of FSI\*ce is motivated by advantages mainly for the developer, but also for a user of it. In addition, a reordering of some functionality and cleaning up of the module structure is essential to enable reasonable further development at all.

**Advantages for the developer** Having all necessary source code files in one project frame enables the use of special features found in modern IDEs (Integrated Development Environments). This is not true for people working with simple text editors, but only for those using environments like Eclipse, KDevelop or similar. These environments offer

cross references to classes and types used; one can jump from the place a function is called directly to its definition, or search for all places the function appears. Proposals to complete a started identifier name are made automatically. All those features are no must to develop a code, but they come in more and more handy, the bigger the project gets and are especially helpful to speed up the understanding of the relations between different classes and functions.

Another important point motivating a new structure are the dependencies of the modules of *FSI\*ce* among each other. *FSIcom* uses *FSImesh* to implement its functionality, for example. If only the implementation of the functionality of *FSImesh* changes, and not the interface, it is not a problem for *FSIcom*, as long as *FSImesh* is linked dynamically (for more information about static and dynamic linking cf. Remark 2.2). But as soon as the interface changes<sup>3</sup>, all modules depending on the changed one must be recompiled, too.

**Remark 2.2.** Static and dynamic linking of libraries

A library contains some closed set of functionality intended to be used by other programs. For a program, in order to make use of a library, it must link it, i.e. include its functionality. There are two forms of linking available at the moment. The first and older form is called static linking. Linking a static library is done during compile time of the program. The program's binary (or executable) simply is enhanced by the functionality in the library. Static linking has two major disadvantages. One occurs, when a library is used by several programs. Every program is enlarged by the size of the library, which leads, in the case of a big library, to a high consumption of memory. The second disadvantage manifests, if the functionality of a static library is changed. Then, all programs intending to use the new functionality must be recompiled.

In order to circumvent this problems, dynamically linked libraries were started to be developed in 1964. These libraries are linked during the runtime of a program, which means every time the program is run, the library is loaded. On Windows machines, these libraries have the file extension `.dll` (dynamic-link libraries), under Linux the extension is `.so` (shared objects).

A developer, having to take care of such a system of libraries does not have an easy job, since he always has to remember the dependencies of the different libraries involved. In a single project frame, these dependencies can be tracked automatically by so-called "targets" in Makefiles. The information, that a library depends on another one, is given only once and the actuality of all libraries installed is guaranteed afterwards.

<sup>3</sup>In C/C++ problems can occur, when header files are modified, which are included by other libraries.

Another issue, related to the GNU build system, is the maintainance of configuration files. The external (or third party) libraries used by the modules of FSI\*ce are mostly the same, such that each library uses a very similar configuration file. A single configure file for all libraries would simplify a change of a library, because only one file has to be taken care of.

**Advantages for the user** A single source code package does not only have advantages for the developer, but also for the user of FSI\*ce. The user must install only one package (which does not necessarily imply the bundling of all functionality in a single installed library) and hence remember only one install routine. The configuration of FSI\*ce for a certain communication technology (sockets or MPI, for example) must be done only once and is propagated to all modules of FSI\*ce.

**Improvements for the software design** Markus Brenk imagined the library structure of FSI\*ce to be threefold, consisting of the libraries `FSImesh`, `FSIcom` and `FSIttools`. `FSImesh` and `FSIcom` have already been explained in the previous section, each of them is forming a layer of the communication architecture (cf. Figure 2.4 on page 11). In addition, `FSIcom` implements the coupling supervisor. `FSIttools` should be a container for additional tools supplied to the user of FSI\*ce, but does not exist as a library in the old structure. The octree functionality, written by Klaus Daubner, would belong into `FSIttools`. `FSIcom` contains actually two ideas, one is the top layer of the communication structure, the other one is the coupling supervisor. The package `FSIviewer` also provides a coupling supervisor featuring a GUI. Since the coupling supervisor plays a very important role, it earns its own package. This package then can offer different implementations of a supervisor - with GUI or without, and can also contain dummy solvers for testing the coupling supervisor. This directly leads to the new structure of FSI\*ce.

### 2.2.4 FSI\*ce Project Frame After Restructuring

The FSI\*ce source code package after restructuring is illustrated in Figure 2.6.

The new project consists of one all enclosing frame, containing two main modules. The module `Libraries` contains the packages `FSImesh`, `FSIcom` and `FSIttools`, with functionality as discussed in the previous sections. Each of these packages is responsible to create one library, that has to be linked by a user of FSI\*ce and by the coupling supervisor.

The module `Supervisor` consists of four packages; `CouplingUnit`, `Batch`, `GUI` and `SolverDummies`. `CouplingUnit` provides the coupling logic and coupling schemes (cf. Section 2.3) to be used when creating a coupling supervisor. `Batch` is responsible for the

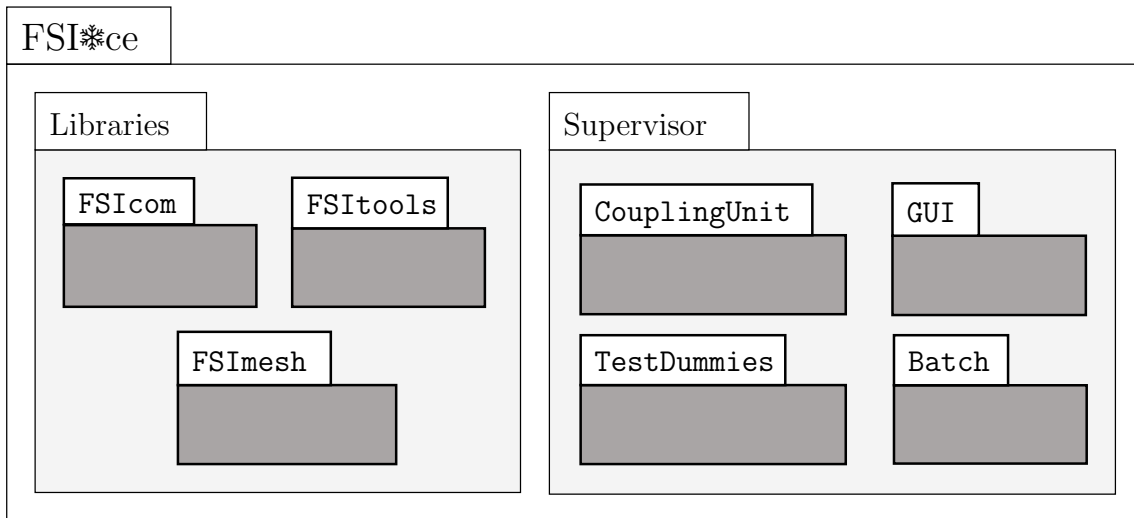


Figure 2.6: FSI\*ce source code package after restructuring. A single source code project contains all sources. The project is subdivided into the modules `Libraries` and `Supervisor`. `Libraries` consists of the packages `FSImesh`, `FSIcom` and `FSIttools`, each creating a library to be used by the coupling supervisor and simulation programs. `Supervisor` contains functionality to create the coupling supervisor and solver dummies. It consists of the packages `Batch`, containing a pure batch-mode running coupling supervisor, `GUI`, containing a coupling supervisor running with GUI, and `SolverDummies` for the dummy solvers.

creation of a pure batch mode executable of a coupling supervisor, which was done in `FSIcom` before. The package `GUI` contains the functionality written by Klaus Daubner to create a coupling supervisor featuring a graphical user interface and enabling the display of coupling relevant quantities. It was one intermediate goal of the thesis, to create a batch substitute for this GUI viewer, since the GUI supervisor does not allow to deactivate the graphical display and is, moreover, closely integrated in some other functionality not desirable to be used in the long run.

A user of FSI\*ce can now create all libraries and the coupling supervisor by only one installation routine, since the configuration scripts of the separated packages are merged into a single one. A developer has everything in one place and does not have to care so much about dependencies any more.<sup>4</sup>

By default, the whole FSI\*ce source code project is in a development mode. This means, that the libraries and the coupling supervisor do only use the project's internal source code, header files and libraries. This can be easily changed to a mode equal to the one of the old project structure, where only installed libraries and header files are used, and hence enable to work with different versions of the same libraries.

<sup>4</sup>The developer of the configuration and makefiles, however, had to improve his skills and knowledge about the GNU build system in a long bothersome process.

## 2.3 Coupling Schemes in Time - Design of a Coupling Unit

In this section, we look at possibilities to couple a two-partitioned system in time and furthermore on the software design of the coupling unit of FSI\*ce, that was developed within this thesis.

### 2.3.1 Ways to Couple

Ideas for this chapter are taken from the work of Carlos A. Felippa on partitioned coupled systems [12], and from the work of Markus Brenk [4]. Although there exists a great variety of schemes on how to couple a partitioned system in time, the basic schemes come down to a few. To partition a system into parts allows a “staggered” solution procedure, where the two equation systems are solved not simultaneously in time, but in series. This reduces at first the costs of solving the overall system of equations, since the growth of costs to solve a system of equations is in general super-linear with the number of unknowns<sup>5</sup> and the overall system can become very ill-conditioned. Everybody would do partitioned fluid-structure interaction, if there was not the downside of degraded stability by the staggered solution procedure. Roughly speaking, the more loosely a system is coupled, the less stable the overall method becomes.

By categorizing coupling schemes according to their strength of preserving the overall system stability, one can divide coupling schemes into weak and strong types. Another common naming scheme is explicit for a weak and implicit for a strong scheme, taken from the terminology of the numerical solutions of ordinary differential equations (ODEs). An introduction to explicit and implicit methods for approximating the solution of ODEs and their characteristics is given in appendix A. It may be remarked, that the intrafield (internal) solution procedure of one simulation program is completely independent from the interfield solution procedure discussed here. The systems may be solved both by an implicit time stepping method with regards to their subfield, but still being coupled together by an explicit scheme, or vice versa.

Figures 2.7, 2.8 and 2.9 illustrate and describe three possible coupling schemes in time. The partitioned systems are denoted by solver  $A$  and solver  $B$ , which would be a fluid and a structure solver in the case of a problem from FSI. The state of a solver’s equation system and interface values at time step  $n$  is denoted by  $(\cdot)_n$ . Each solver uses interface values computed by its coupling partner, and has to compute interface values to be communicated to its coupling partner in turn. Solver  $A$ , for example, works with interface values  $a$ , and computes interface values  $b$  for solver  $B$ . The first scheme (Figure 2.7) is a basic weak scheme with staggered solution procedure. The

---

<sup>5</sup>Gaussian Elimination has costs of order  $O(N^3)$  for example

second scheme (Figure 2.8) is similar, but uses an additional prediction step to increase stability. The last scheme (Figure 2.9) is a typical strong coupling scheme, employing an iteration of the interface values. For this strong scheme, measures for the convergence of the interface values must be used and methods to speed up the convergence are highly valuable. A detailed description of one coupling cycle is added to each Figure.

Another important coupling technique, in some sense orthogonal to the coupling schemes described in the previous paragraph, is called *subcycling*. Subcycles can be performed by each solver independently, and basically mean a series of non-coupled time steps, taking place in-between two “coupled” time steps. Figure 2.10 shows subcycling in combination with a strong coupling scheme. In FSI problems, subcycling is usually performed by the fluid solver, having more restrictive conditions on the time step length than the structure solver.

Finally, the decision which coupling scheme is the best is problem dependent. A simple weak coupling scheme is always the first choice, when stability is no problem. The interface iterations of the strong coupling scheme can be very costly, especially for the fluid part of an FSI simulation with its many unknowns. Hence, a compromise has to be found, with as few costs as possible and satisfying stability.

### 2.3.2 Software Design and Functionality of the New Coupling Unit

In the software design of FSI\*ce, the implementation of the coupling scheme logic should be separated from the simulation tools to be coupled (cf. Section 2.1.1). At the beginning of this work, the coupling communication logic was implemented in the package FSIcom (cf. Section 2.2.2) for both, the coupling supervisor and the simulation software to be coupled. Since the coupling supervisor has now its own package (cf. Section 2.2.4) in the project frame of FSI\*ce, the coupling functionality of the supervisor was separated from FSIcom and a new, object-oriented software frame was created for it. A source code frame that supports not only FSI simulations, but in a wider frame any arbitrary multiphysics application, based on a partitioned simulation approach was set up.

Figure 2.11 shows the main classes realizing the logic of the coupling unit. The main coupling logic is implemented in the class `CouplingScheme`, which decides on the coupling steps to be performed. `CouplingScheme` uses the class `Participant` to represent the participating solvers of the coupled simulation and to store their state of computation.

Actual use of the coupling unit is made when creating a coupling supervisor. In Listing 2.1, an example for the main part of a coupling supervisor is given. With help of this example, we will look at the functionality the coupling unit is able to provide. In a first phase, the frame of the coupled simulation has to be set up. A solver with name “Fluid Solver”, further called fluid solver, is added in line 3. The first solver

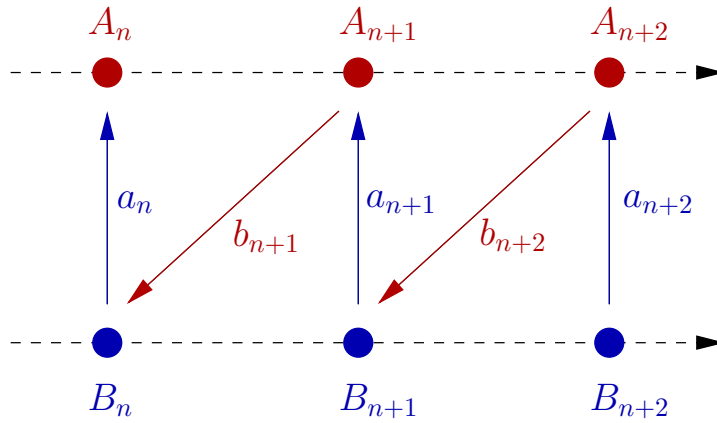


Figure 2.7: **Basic weak coupling scheme with staggered solution procedure.** (The dashed lines indicate the time axis for each solver). The solution procedure starts at time  $t_n$  with solver  $B$  sending its initial interface values  $a_n$  to solver  $A$  (this step can be omitted, if solver  $A$  is supplied with proper initial interface values). Solver  $A$  computes its next time step  $t_{n+1}$  and sends the obtained interface values  $b_{n+1}$  to solver  $B$ , which in turn advances to the next time step.

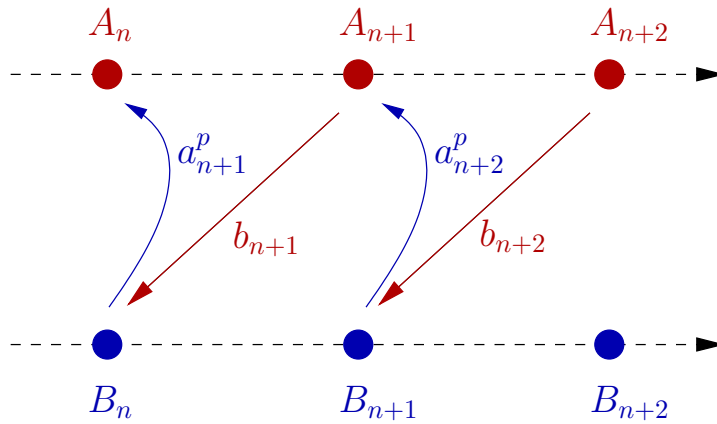


Figure 2.8: **Weak coupling scheme with predictor and staggered solution procedure.** (The dashed lines indicate the time axis for each solver). The solution procedure starts with solver  $B$ , which has to compute a prediction  $a_{n+1}^p$  for the interface values of the next time step, and send it to solver  $A$ . Solver  $A$  computes its next time step  $t_{n+1}$  and sends the obtained interface values  $b_{n+1}$  to solver  $B$ , which in turn computes its next time step.

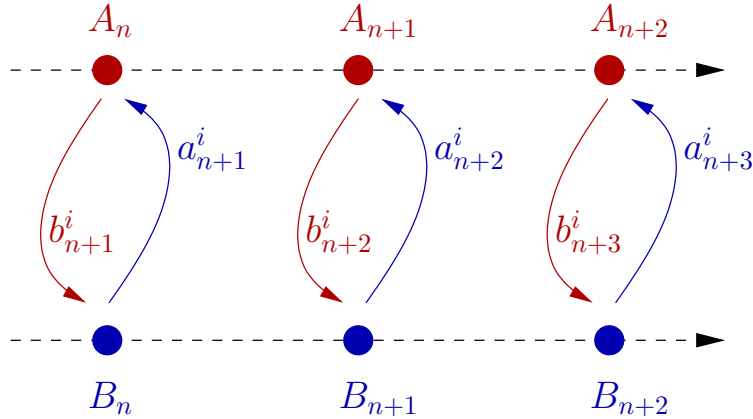


Figure 2.9: **Strong coupling scheme with interface iterations.** (The dashed lines indicate the time axis for each solver). The solution procedure starts with solver  $A$  (but could equally start with solver  $B$ ) at time  $t_n$ . Solver  $A$  computes a first prediction  $b_{n+1}^1$  for solver  $B$ 's interface values of the next time step and sends it to solver  $B$ .  $B$  uses the obtained prediction to compute itself a prediction  $a_{n+1}^1$  for solver  $A$ 's interface values and returns it to solver  $A$ . This procedure continues (next would be solver  $A$  to compute  $b_{n+1}^2$ ), with both solvers always using the most current interface values, until convergence of the interface values is achieved. Then, solver  $A$  and solver  $B$  can advance simultaneously to time step  $t_{n+1}$ .

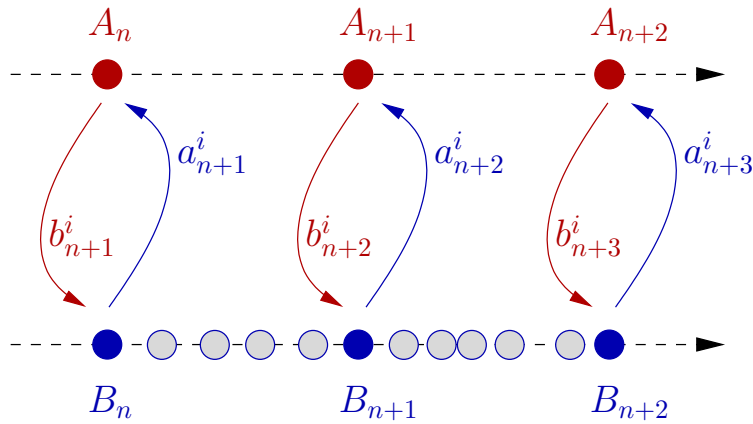


Figure 2.10: **Subcycling solver  $B$  combined with a strong coupling scheme.** (The dashed lines indicate the time axis for each solver). The solution procedure follows the description of Figure 2.9, with a difference for solver  $B$  when advancing to the next time step. Then, solver  $B$  computes non-coupled time steps (indicated by circles filled with lighter color) always using the interface values obtained by  $A$  at time  $t_n$  until it reaches time  $t_{n+1}$ . Only then a new interface iteration starts. Subcycling can be equally combined with any other of the discussed coupling schemes.

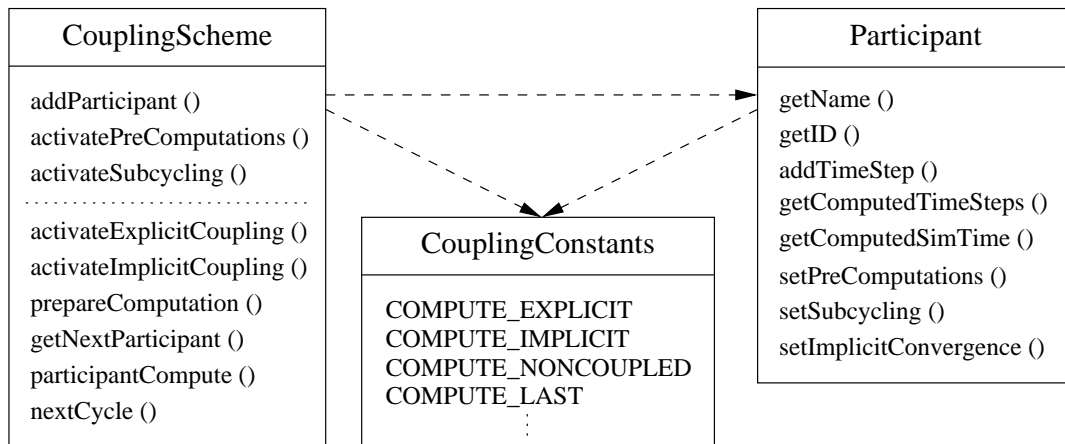


Figure 2.11: Main part of coupling unit software design, used in coupling supervisor. The class `CouplingScheme` implements the logic that decides on the coupling steps to be performed. The class `Participant` serves as state encapsulator for each solver taking part in the coupled simulation. The implemented functionality can serve as basis for arbitrary multiphysics simulations, allowing more than two participating solvers.

added is also the solver starting the coupled simulation. In line 4, pre-computations are assigned to the fluid solver. The concept of pre-computations enables to create a physically valid initial frame for the simulation. The fluid solver is advised to compute at least 15 seconds of simulation time, before the actual coupled simulation starts. In line 5, subcycling (cf. previous section 2.3), with at least 10 intermediate steps, is assigned to the fluid solver. A minimum amount of subcycling steps might be necessary to regain a physically reasonable state in the flow field after a change of the structure geometry. The length of the subcycle time steps for the fluid solver is decided dynamically, which ensures uniform time step lengths. A second solver, called “Solid Solver” is added to the simulation, without any special properties, in line 7. Then, the overall coupling scheme to be performed during the simulation is selected to be implicit (line 8). Within this thesis, only a basic weak and an implicit coupling scheme are offered as described in the previous section. The set-up phase is closed by printing a detailed coupling status (line 10) and preparing the coupling unit to start the coupling cycles in line 11. This set-up phase will be triggered from an XML-file in future, offering an easy and central configuration of the coupled simulation to a user of FSI\*ce.

The actual coupling cycles follow in line 13 to 18. An outer loop iterates the coupling cycles, while an inner loop iterates the participants active in a cycle. All further logic is hidden within the class `CouplingScheme`.

```

1 CouplingScheme cplScheme;
2
3 cplScheme.addParticipant (FLUID_ID, "Fluid Solver");
4 cplScheme.activatePreComputations (FLUID_RANK, 0, 15.0);
5 cplScheme.activateSubcycling (FLUID_RANK, 10);
6
7 cplScheme.addParticipant (SOLID_ID, "Solid Solver");
8
9 cplScheme.activateImplicitCoupling ();
10 cplScheme.printCouplingStatus ();
11 cplScheme.prepareComputation ();
12
13 do {
14     while ( cplScheme.getNextParticipant(id) )
15         cplScheme.participantCompute (id, couplingMesh);
16 } while ( cplScheme.nextCycle() )

```

Listing 2.1: Pseudo code of coupling supervisor

## 2.4 Modifications on the Application Programming Interface

A user of FSI\*ce wants to be able to integrate the coupling interface functionality with only a few intrusions into his solver’s code. This interface, usually referred to as “Application Programming Interface” (API), is from the developer’s point of view the outward door for the functionality of FSI\*ce. This door must be designed with great care, because every change to its shape does affect all users of FSI\*ce, requiring them to adapt their codes, too. If the API of FSI\*ce is designed well, changes and extensions of functionality made later can be simply transmitted to a user by updating his FSI\*ce libraries.

The design of the coupling interface created by Markus Brenk in [4] is strongly oriented around the commercial coupling tool MPCCI [1], which in turn is similar to the standard communication interface for parallel programming, MPI ([17], [18]). This orientation is very useful, since most people developing simulation tools will be familiar with MPI, too. Listing 2.2 shows the API of FSI\*ce, as designed by Markus Brenk, in a simplified version. It consists of functions to initialize and to finalize the coupled simulation (lines 1, 2), and to send and to receive the coupling interface data (lines 3, 4).

During the work on a coupling unit for FSI\*ce, the need to still improve the coupling API and better adapt it to a more powerful coupling supervisor emerged. The coupling interface presented in listing 2.2 transmits, besides the indispensable coupling interface values, only a flag from the supervisor to the coupled solvers. The flag determines

---

```

1  int Fsi_Init ();
2  int Fsi_Finalize ();
3  int Fsi_Recv_quantity ();
4  int Fsi_Send_quantity ();

```

---

Listing 2.2: FSI\*ce API as designed by Markus Brenk. Function parameters are omitted.

whether the simulation shall still continue to run, and whether implicit iterations are to be performed by the solvers. The logic of subcycling, for example, must be implemented by the solvers themselves. It is not possible to pack this information into the same single flag, since implicit as well as explicit time stepping can be combined with subcycles. This leaves the idea of a coupling supervisor, encapsulating all coupling logic and acting as main control unit to some extent unfinished. Thus, a new interface was devised, that inquires more information from a coupled solver, but in turn offers advanced coupling possibilities built into the coupling supervisor.

Listing 2.3 shows the new coupling API. A first call to receive a coupling mesh with computed values is now integrated in the function `FSI_Init`. This call is necessary for the coupled solver not starting the coupled simulation, but waiting for the first computed interface values (In a typical FSI simulation, this will be the structure solver.). In the old coupling API, this first call to receive valid interface values had to be implemented by the user himself into his solver. The main change of the new interface is that explicit send and receive calls are hidden from the user of FSI\*ce and offer him only a single interface function `FSI_Data_exchange`. Within this function, several bi-directional communication calls are taking place in-between the coupling supervisor and a coupled solver. The coupling interface mesh is now only exchanged if necessary, which saves communication overhead. The solver transmits the length of its computed time step to the supervisor and gets back an upper limit for the computation of its next time step. This upper limit for the time step length enables to exactly close a subcycle period or preliminary computations of a solver.

The additional interface functions of line 5, 6 and 7 provide access for a solver to the information received from the coupling supervisor. While, in a minimal effort case, only the function `FSI_Is_running` is really mandatory to be used, the function `FSI_Is_implicit_converged` is necessary when performing an implicitly coupled simulation, and the function `FSI_Is_new_interface_values` can save a solver from unnecessarily updating its interface values with the same old values.

Listing 2.4 shows the new coupling API integrated in a reference solver code. In this variant of a solver code, the update of the interface values from the coupling mesh takes place in the beginning of the time stepping loop. It is possible to shift this update to the end of the time stepping loop without any consequences, if the solver is always starting the coupled simulation. This could be true for a fluid solver in an FSI simulation. However, a more flexible setup with update of the coupling interface values at the end of

```
1 void FSI_Init ();
2 void FSI_Data_Exchange ( timeStepLength );
3 void FSI_Finalize ();
4
5 int FSI_Is_new_interface_values ();
6 int FSI_Is_implicit_converged ();
7 int FSI_Is_running ();
```

Listing 2.3: New FSI\*ce API, allowing the operation of an advanced coupling supervisor, with more control possibilities and hence, greater coupling flexibilities. Function parameters are mostly omitted.

the time stepping loop is presented in listing 2.5, where an additional update of coupling interface values is performed prior to the time stepping loop.

---

```

1  FSI_Init (FSI_Mesh, dt);
2  while ( FSI_Is_running() )
3      if ( FSI_Is_new_interface_values() )
4          Update solution at coupling interface
5      end
6      Set time step for computation
7      Compute values of next time step
8      Write coupling interface values into FSI_Mesh
9      FSI_Data_exchange (FSI_Mesh, dt);
10     if ( FSI_Is_implicit_converged() )
11         Store computed values of next time step
12     end
13 end
14 FSI_Finalize ();

```

---

Listing 2.4: New coupling API integrated into reference solver’s code of variant one. This variant of a solver code updates its coupling interface values at the beginning of the time stepping loop.

---

```

1  FSI_Init (FSI_Mesh, dt);
2  if ( FSI_Is_new_interface_values() )
3      Update solution at coupling interface
4  end
5  while ( FSI_Is_running() )
6      Set time step for computation
7      Compute values of next time step
8      Write coupling interface values into FSI_Mesh
9      FSI_Data_exchange (FSI_Mesh, dt);
10     if ( FSI_Is_implicit_converged() )
11         Store computed values of next time step
12     end
13     if ( FSI_Is_new_interface_values() )
14         Update solution at coupling interface
15     end
16 end
17 FSI_Finalize ();

```

---

Listing 2.5: New coupling API integrated into reference solver code of variant two. This variant of a solver code updates its coupling interface values at the end of the time stepping loop. To be still able to start a coupled simulation with receiving already computed interface values, an additional update must be placed before the time stepping loop, after a call to *FSI\_Init*.

## 3 About Fluids and Structures

Before describing the fluid and structure simulation tools used within this thesis in Chapter 4, a short derivation of the underlying physical models for fluid flow (cf. Section 3.1), structure mechanics (cf. Section 3.2) and the coupling surface equations (cf. Section 3.3) of these two domains is given here.

The notation used to write down the laws of fluid and structure mechanics is the tensor notation in combination with the Einstein summation convention. This notation avoids clutter with vector arrows or underlines and gives a more intuitive understanding, in particular of larger equations. However, for the reader disliking tensor notation, the main equations derived are also given in non-tensor notation. For that, underbars are used to indicate vectors and double underbars to indicate tensors. A short introduction to tensor notation is given in Appendix B.

### 3.1 Physical Model for Fluids

In this first section, we will look at the fundamental physical laws describing fluid motion. We limit our investigations to a very special class of fluids, namely incompressible (cf. Remark 3.1) Newtonian (cf. Remark 3.2) fluids with constant properties. After introducing the basic principles of fluid flow, the conservation laws of mass and momentum are derived. This leads to the main set of equations describing the fluids considered in this work, namely the Navier-Stokes equations. We will see how the Navier-Stokes equations are obtained as a specialization of the conservation laws for mass and momentum and, finally, look at the dimensionless form of them.

The reader already familiar with, or not interested in these derivations, finds the resulting dimensionless Navier-Stokes equations (3.41) to (3.44) in Section 3.1.7. This form of the equation set will be used in all following chapters.

**Remark 3.1.** Incompressible fluids

In general, a fluid is considered to be incompressible, if it has constant density. But also compressible fluids with low Mach number, i.e. if the speed of sound is much higher than the velocity of the fluid itself, can be considered to be incompressible. Further more, fluids with relatively slow changing density over distance can be assumed to be incompressible, despite their density is not constant on the large scale. Examples for such a kind of a fluid are the oceanic water, or the earth's atmosphere.

**Remark 3.2.** Newtonian fluids

A Newtonian fluid exhibits a linear relationship of the rate of deformation  $\partial v/\partial x$  with respect to the shear stress  $\tau$  applied to deform the fluid. This property can be expressed by the following equation

$$\tau = \mu \frac{\partial v}{\partial x},$$

with  $\mu$  as scale for the (shear) viscosity (for constant property fluids constant in time and space),  $v$  as fluid velocity and  $x$  as position.

### 3.1.1 The Continuum Principle

To describe fluid flow with tools of modern mathematics in the form of partial differential equations, we have to introduce the principle of the continuum. We know that, in reality, fluids are made up of molecules, with each molecule moving around in its own, at usual temperatures highly irregular way. However, we want to assign macroscopic attributes such as temperature, pressure or density to the fluid field. Thus, we introduce the so-called fluid elements as basic elements, where each fluid element consists of an aggregation of fluid molecules as shown in figure 3.1.

We can assign attributes to these fluid elements in a meaningful way by averaging over the molecules' properties. Then, we can describe the current state of a flow by position  $x$  and the time  $t$  as independent variables. To make a fluid accessible by mathematical calculus we assume that it is composed of sufficiently, dense packed fluid elements<sup>1</sup>, such that we can form derivatives with respect to the field variables and time. The validity

---

<sup>1</sup>To be mathematically correct, we would need infinitely many, infinitely small, infinitely dense packed fluid elements. However, since we only compute numerical approximations of the exact fluid state anyhow, this mathematical incorrectness is not of relevance for almost all cases.

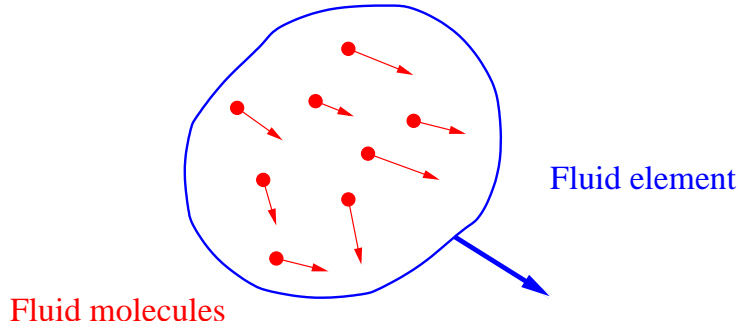


Figure 3.1: Fluid element composed of fluid molecules, with arrows indicating the directed velocity.

of this hypothesis is described quantitatively by the *Knudson number*

$$Kn = \frac{\lambda}{l}, \quad (3.1)$$

which is formed by the ratio of the two characteristic quantities *mean free path*  $\lambda$  of a fluid molecule and *smallest geometric length scale*  $l$  of the considered flow domain. If  $Kn \ll 10^{-3}$  is valid, the continuum approach is considered to be appropriate.

### 3.1.2 Lagrangian and Eulerian View

The mathematical formulation of physical phenomena depends heavily on the frame of reference taken by the observer<sup>2</sup>. Two possible views an observer may take are the Lagrangian and the Eulerian view. The Lagrangian view fixes the frame of reference to the object observed. The observer is bound to the object of interest and follows its movements. The Lagrangian approach is usually taken when describing the behavior of structures in mechanics. In the Eulerian view, the observer takes a step back and uses an external reference frame. He looks at fixed locations of interest and sees the so-called field variables changing there, influenced by objects passing through. The Eulerian view is typically used in fluid dynamics, where one does not only consider a single fluid element, as described in the previous Section 3.1.1, but a whole region filled with objects, no longer distinguishable. Figure 3.2 illustrates the Lagrangian and Eulerian view, with a Cartesian coordinate system as reference point.

The position of a fluid element in the Lagrangian view is given by

$$X = X(t; X_0), \quad (3.2)$$

---

<sup>2</sup>A well known scientific example for the influence of different points of view is the observation of the trajectories of the planets in our solar system. After Nikolaus Kopernikus in the 16th century had come to the conclusion, that the sun must be the center of our solar system, he could obtain the circle-like trajectories of the planets around the sun. The trajectories observed before him were of course not wrong, but to fix the reference system to the sun instead of the earth finally allowed it to gain deeper astronomical insights.

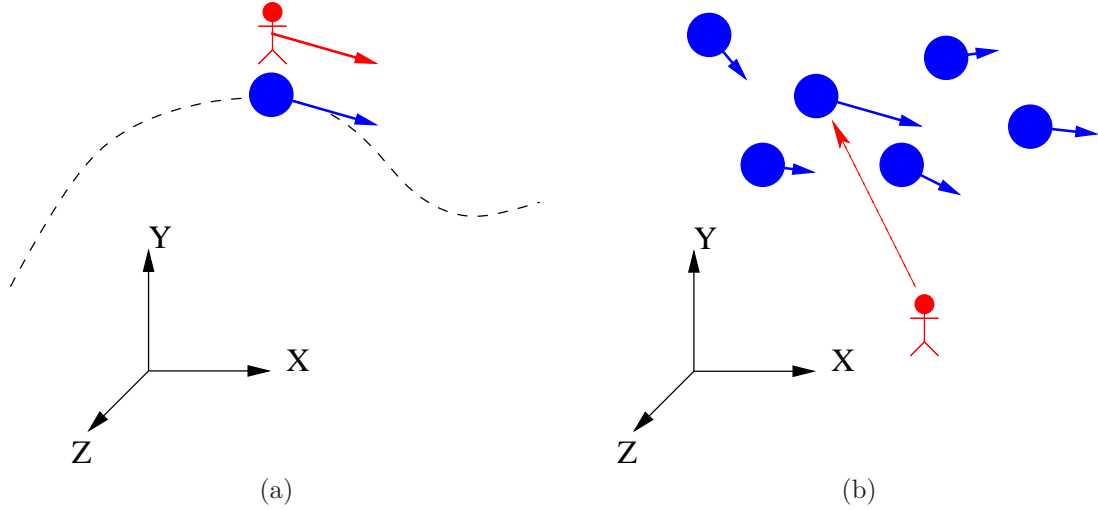


Figure 3.2: Lagrangian (a) and Eulerian (b) points of view. In the Lagrangian point of view, the observer fixes his frame of reference to the object observed. In the Eulerian point of view, the observer steps back and looks at the properties of objects passing through fixed points of interest.

depending on time  $t$  and the initial position  $X_0$  of the fluid element.  $X_0$  is uniquely characterizing the fluid element and is defined at time  $t_0$  by

$$X(t_0, X_0) = X_0. \quad (3.3)$$

The velocity of a fluid element is then given by the change of its position in time

$$\frac{\partial}{\partial t} X(t; X_0) = V(t; X_0). \quad (3.4)$$

The connection of Lagrangian and Eulerian view is established by considering the influence of fluid elements, described by the Lagrangian viewpoint, on the property of a point observed in Eulerian view. The velocity  $v$  observed at the fixed point  $x$  depends on the fluid elements passing through this point at times  $t$ , such that

$$v(x, t) = v(X(t; X_0), t). \quad (3.5)$$

Then, the total change of velocity  $v$  in time is formed by the chain rule

$$\begin{aligned} \frac{dv_i}{dt} &= \frac{\partial v_i}{\partial t} + \frac{\partial X_j}{\partial t} \frac{\partial v_i}{\partial x_j} \\ &= \frac{\partial v_i}{\partial t} + V_j \frac{\partial v_i}{\partial x_j}, \end{aligned} \quad (3.6)$$

and since for all fluid particles with  $x = X(t; X_0)$  the equality  $v(x, t) = V(t; X_0)$  holds, we can write (3.6) as

$$\frac{dv_i}{dt} = \frac{\partial v_i}{\partial t} + v_j \frac{\partial v_i}{\partial x_j}. \quad (3.7)$$

We can generalize (3.7) from the velocity  $v$  to any property observable in the Eulerian field and obtain the *material* or *substantial* derivative of that property at position  $x$  and time  $t$  by

$$\frac{d}{dt} = \frac{\partial}{\partial t} + v_i \frac{\partial}{\partial x_i}. \quad (3.8)$$

In non-tensor notation, (3.8) reads

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \underline{v} \cdot \nabla. \quad (3.9)$$

### 3.1.3 Reynolds Transport Theorem

In order to derive the mathematical formulation of the conservation laws of fluid mechanics, we need to leave the consideration of single fluid elements and switch to so-called control volumes, consisting of dense aggregations of fluid elements. The Reynolds transport theorem relates the Lagrangian and Eulerian view, as the material derivative (3.8) does, but for integral formulations.

Considering the change of the integral of a property  $\beta$  over a control volume  $V$  depending on time yields the Reynolds transport theorem for property  $\beta$

$$\frac{d}{dt} \int_{V(t)} \beta dV = \int_{V(t)} \frac{\partial \beta}{\partial t} dV + \int_{S(t)} \beta v_i n_i dS. \quad (3.10)$$

Here,  $S$  is the surface of  $V$  and  $n$  the normal vector of the control volume's surface, pointing outwards. The first term on the right side describes the change of property  $\beta$  within the control volume, while the second term describes the in- and outflow of property  $\beta$  over the surface of  $V$ . It may be remarked, that despite  $\beta$  is written as scalar here, the Reynolds transport theorem is equally valid for vectorial properties or tensors of arbitrary order. The time dependency of  $V$  will be omitted from now on, for reasons of clarity.

Transforming the surface integral into a volume integral by applying the Gaussian integration rule, we obtain

$$\frac{d}{dt} \int_V \beta dV = \int_V \left( \frac{\partial \beta}{\partial t} + \frac{\partial(\beta v_i)}{\partial x_i} \right) dV, \quad (3.11)$$

which reads in non-tensor notation as

$$\frac{d}{dt} \int_V \beta dV = \int_V \left( \frac{\partial \beta}{\partial t} + \nabla \cdot (\beta \underline{v}) \right) dV. \quad (3.12)$$

### 3.1.4 Conservation of Mass

In a given system without any sources or sinks of mass the total mass must be conserved and, hence, remain constant. The mass of a system is determined by the integral of density  $\rho$  over the system's volume  $V$

$$m(V) = \int_V \rho dV. \quad (3.13)$$

The change of mass must be zero

$$\frac{dm}{dt} = \frac{d}{dt} \int_V \rho dV = 0. \quad (3.14)$$

Applying the Reynolds transport theorem (3.11) to (3.14) yields

$$\frac{dm}{dt} = \int_V \left( \frac{\partial \rho}{\partial t} + \frac{\partial(\rho v_i)}{\partial x_i} \right) dV = 0. \quad (3.15)$$

Since the conservation of mass must be valid for any volume, it must also be valid for infinitesimally small ones, such that the integral can vanish

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho v_i)}{\partial x_i} = 0. \quad (3.16)$$

Equation 3.16 is called *continuity equation* in differential form. For incompressible fluids with constant density, as considered in this work, the continuity equation reduces to

$$\frac{\partial v_i}{\partial x_i} = 0. \quad (3.17)$$

In non-tensor notation, (3.17) reads

$$\nabla \cdot \underline{v} = 0. \quad (3.18)$$

### 3.1.5 Conservation of Momentum

The conservation of momentum is another important conservation law in fluid mechanics. It is based on Newton's second law, which states that the change of an object's momentum (which is mass  $m$  times velocity  $v$ ) yields a force  $F$

$$F_i = \frac{d(mv_i)}{dt}. \quad (3.19)$$

Applying this relation in the form of a conservation law over a control volume  $V$  leads to

$$F_i = \frac{d}{dt} \int_V \rho v_i dV. \quad (3.20)$$

We consider two types of forces acting on our control volume. The first type of force is called volume force and is given by

$$F_{V,i} = \int_V \rho f_i dV, \quad (3.21)$$

where  $f$  is called distributed force. A typical instance of a volume force is gravity. The second type of forces under consideration are forces acting on the surface of our control volume

$$F_{S,i} = \int_S \sigma_{ij} n_j dS, \quad (3.22)$$

where the surface stress tensor  $\sigma$  is given by

$$\sigma_{ij} = -p\delta_{ij} + \tau_{ij} = \begin{pmatrix} -p + \tau_{11} & \tau_{12} & \tau_{13} \\ \tau_{21} & -p + \tau_{22} & \tau_{23} \\ \tau_{31} & \tau_{32} & -p + \tau_{33} \end{pmatrix}. \quad (3.23)$$

It is composed of the pressure  $p$  multiplied by the unit tensor  $\delta$  (which has ones at the diagonal and is zero elsewhere) and summed up with the stress tensor  $\tau$ . Tensor  $\tau$  consists of diagonal components, called normal stresses, and non-diagonal components, called shear stresses. Because of the equilibrium of the moments of torque acting on a fluid element, the stress tensor  $\tau$  must be symmetric, i.e.  $\tau_{ij} = \tau_{ji}$ .

Now, we can take equation (3.20) and expand it by the Reynolds transport theorem. Doing so, we obtain

$$F_i = \int_V \left( \frac{\partial(\rho v_i)}{\partial t} + \frac{\partial(\rho v_i v_j)}{\partial x_j} \right) dV. \quad (3.24)$$

The change of momentum must be equal to the external forces applied, i.e. volume and surface forces, resulting in

$$\int_V \left( \frac{\partial(\rho v_i)}{\partial t} + \frac{\partial(\rho v_i v_j)}{\partial x_j} \right) dV = - \int_S p n_i dS + \int_S \tau_{ij} n_j dS + \int_V \rho f_i dV. \quad (3.25)$$

After rewriting the surface integrals into volume integrals, (3.25) reads

$$\int_V \left( \frac{\partial(\rho v_i)}{\partial t} + \frac{\partial(\rho v_i v_j)}{\partial x_j} \right) dV = \int_V \left( -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho f_i \right) dV. \quad (3.26)$$

Considering infinitesimally small control volumes, we obtain the differential form of the momentum equation (3.26)

$$\frac{\partial(\rho v_i)}{\partial t} + \frac{\partial(\rho v_i v_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho f_i. \quad (3.27)$$

By applying the continuity equation (3.16) or (3.17) to the momentum equation (3.27), we arrive at the final form of the momentum equation

$$\rho \frac{\partial v_i}{\partial t} + \rho v_j \frac{\partial v_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho f_i, \quad (3.28)$$

In non-tensor notation, the momentum equation reads

$$\rho \frac{\partial \underline{v}}{\partial t} + \rho(\underline{v} \cdot \nabla)\underline{v} = -\nabla p + \nabla \underline{\underline{\tau}} + \rho \underline{f}. \quad (3.29)$$

### 3.1.6 The Navier-Stokes Equations for Incompressible Flows

We have derived the equation set capable of describing the behavior of incompressible flows in the two previous sections 3.1.4 and 3.1.5. In order to have a closed set of equations, i.e. a set of equations providing a solution for all its unknowns, we need to specify an ansatz for the stress tensor  $\tau$ . Stokes generalized the ansatz for stresses in Newtonian fluids under the following assumptions

1. Conservation of torque within every fluid element, which yields the symmetry of the stress tensor.
2. Isotropic fluid properties, i.e. same behavior in every direction.
3. For a resting fluid, it must be true that  $\sigma_{ij} = -p\delta_{ij}$ , i.e. normal and shear stress cancel out.

The generalized Newtonian ansatz for the stress tensor  $\tau$ , which is part of the surface stress tensor  $\sigma$  (cf. equation (3.23)) reads

$$\tau_{ij} = \mu \left( \frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} - \frac{2}{3} \delta_{ij} \frac{\partial v_k}{\partial x_k} \right) + \mu' \delta_{ij} \frac{\partial v_k}{\partial x_k}. \quad (3.30)$$

Variable  $\mu$  is called the (shear) viscosity of the fluid, which is its property to offer resistance against shear stresses.  $\mu'$  is called volume viscosity and represents the fluid's resistance against compression. Stoke's hypothesis states that

$$\mu' = 0, \quad (3.31)$$

which is taken to be granted for all further considerations. In addition, for incompressible, constant property fluids, the following simplifications are possible:

$$\frac{\partial v_k}{\partial x_k} = 0, \quad (3.32)$$

which is basically the continuity equation for incompressible flows (3.17).

$$\frac{\partial \mu}{\partial x_j} = 0, \quad (3.33)$$

which must be true for a constant property flow. By applying Stoke's hypothesis (3.31) and the simplification (3.32) and (3.33) to the Newtonian stress ansatz (3.30) and inserting the simplified stress tensor  $\tau$  into the momentum equations (3.28), we obtain the

momentum equations in differential form, for constant property Newtonian fluids under Stoke's hypothesis

$$\rho \frac{\partial v_i}{\partial t} + \rho v_j \frac{\partial v_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \mu \frac{\partial^2 v_i}{\partial x_j \partial x_j} + \rho f_i. \quad (3.34)$$

In non-tensor notation, the momentum equations read

$$\rho \frac{\partial \underline{v}}{\partial t} + \rho (\underline{v} \cdot \nabla) \underline{v} = -\nabla p + \mu \Delta \underline{v} + \rho \underline{f}. \quad (3.35)$$

The second term on the left side of the momentum equations is called convective term. It models the influence of inertia of the fluid elements. The second term on the right side is called diffusive term and is responsible for modeling the friction of the fluid, resulting in diffusive transport of velocity.

Together with the continuity equation for incompressible flows (3.17)

$$\frac{\partial v_i}{\partial x_i} = 0,$$

the momentum equations form the *Navier-Stokes equations*, a closed system of equations, capable of describing all phenomena of a flow with mentioned properties.

### 3.1.7 Rendering the Navier-Stokes Equations Dimensionless

The Navier-Stokes equations comprise certain similarity properties (or invariance properties) with regards to its variables. One important similarity property is the Reynolds-Number similarity. This property states, that two flows can be similar, although their flow domain, velocities, density and viscosity differ from each other. It enables to build small scale models, e.g. for wind tunnel experiments, that show the same behavior as the real objects. To make this property visible, a proper scaling of the Navier-Stokes equations has to be introduced first. The Navier-Stokes equations have to be rendered dimensionless and all characterizing quantities have to be gathered, which results in the Reynolds-Number  $Re$  (and Froud number  $Fr$ ). Understanding the scaling of the Navier-Stokes equations is important to understand how to rescale values obtained by a flow simulation using the dimensionless equations. Thus, we will go through the procedure of rendering the Navier-Stokes equations dimensionless here.

We start off with the momentum equations (3.34), as derived in the previous section

$$\rho \frac{\partial v_i}{\partial t} + \rho v_j \frac{\partial v_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \mu \frac{\partial^2 v_i}{\partial x_j \partial x_j} + \rho f_i.$$

We divide the momentum equations by  $\rho$  and obtain

$$\frac{\partial v_i}{\partial t} + v_j \frac{\partial v_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 v_i}{\partial x_j \partial x_j} + f_i, \quad (3.36)$$

with the kinematic viscosity  $\nu := \mu/\rho$ . In order to render the Navier-Stokes equations dimensionless, we introduce dimensionless variables denoted by  $(\cdot)^*$  and characteristic variables denoted by  $(\cdot)_\infty$ , which have the relations

$$\begin{aligned} x^* &:= \frac{x}{L_\infty} && \text{dimensionless coordinate,} \\ v^* &:= \frac{v}{V_\infty} && \text{dimensionless velocity,} \\ p^* &:= (p - P_\infty) \frac{1}{\rho V_\infty^2} && \text{dimensionless pressure,} \\ t^* &:= \frac{t}{t_\infty} = \frac{t}{\frac{L_\infty}{V_\infty}} = t \frac{V_\infty}{L_\infty} && \text{dimensionless time.} \end{aligned}$$

The characteristic time  $t_\infty$  is actually predetermined by the free choice of a characteristic length  $L_\infty$  and velocity  $V_\infty$ . It is also possible to choose the time as degree of freedom and predetermine one of the other characteristic variables. Inserting the characteristic relations into the momentum equations (3.36) yields

$$\frac{V_\infty^2}{L_\infty} \frac{\partial v_i^*}{\partial t^*} + \frac{V_\infty^2}{L_\infty} v_j^* \frac{\partial v_i^*}{\partial x_j^*} = -\frac{V_\infty^2}{L_\infty} \frac{\partial p^*}{\partial x_i^*} + \frac{V_\infty}{L_\infty^2} \nu \frac{\partial^2 v_i^*}{\partial x_j^* \partial x_j^*} + f_i. \quad (3.37)$$

By multiplying equation (3.37) with  $L_\infty/V_\infty^2$  we get

$$\frac{\partial v_i^*}{\partial t^*} + v_j^* \frac{\partial v_i^*}{\partial x_j^*} = -\frac{\partial p^*}{\partial x_i^*} + \frac{\nu}{V_\infty L_\infty} \frac{\partial^2 v_i^*}{\partial x_j^* \partial x_j^*} + \frac{L_\infty}{V_\infty^2} f_i. \quad (3.38)$$

The remaining characteristic quantities are summarized to the dimensionless Reynolds number

$$Re := \frac{V_\infty L_\infty}{\nu} = \frac{V_\infty L_\infty \rho}{\mu}, \quad (3.39)$$

and Froud number

$$Fr^2 := \frac{V_\infty^2}{L_\infty}, \quad (3.40)$$

The Reynolds number modulates the influence of the convective and diffusive terms of the Navier-Stokes equation, while the Froud number modulates the influence of the volume force term.

Omitting the  $*$  from the dimensionless variables, one can write the *dimensionless momentum equations* in their final form as

$$\frac{\partial v_i}{\partial t} + v_j \frac{\partial v_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re} \frac{\partial^2 v_i}{\partial x_j \partial x_j} + \frac{1}{Fr^2} f_i, \quad (3.41)$$

or, in non-tensor notation as

$$\frac{\partial \underline{v}}{\partial t} + (\underline{v} \cdot \nabla) \underline{v} = -\nabla p + \frac{1}{Re} \Delta \underline{v} + \frac{1}{Fr^2} \underline{f}. \quad (3.42)$$

The continuity equation (3.17) can be used as dimensionless equation by simply substituting its variables with their corresponding dimensionless version and dividing by the proper inverse factor to cancel out the characteristic variables. The *continuity equation in dimensionless form* reads

$$\frac{\partial v_i}{\partial x_i} = 0, \quad (3.43)$$

which is in non-tensor notation

$$\nabla \cdot \underline{v} = 0. \quad (3.44)$$

## 3.2 Physical Model for Structures

The description of structures originates, as fluid dynamics does, in continuum mechanics. The main concepts of the continuum description are very similar to those of fluids (cf. Section 3.1.1), with solid particles instead of fluid elements. We will not go into as much detail as with the description of the fluid mechanics laws, since investigations into the structure part of the fluid-structure interaction problem played only a minor part during my work on this thesis. In the following sections, we will review some fundamental concepts of structural mechanics, the strain- and stress tensor, kinetics and the relationship of strain and stress. The main source for these sections was the work of Markus Brenk [4]. Further and more detailed descriptions of the fundamentals of mechanics can be found in [3] and [22], for example.

### 3.2.1 Strain Tensor

Strain is a measure on how heavily a body is deformed and is one important means to describe the state of a rigid body. Mathematically, the strain of a rigid body can be described in the form of a tensor, namely by the *strain tensor*. To derive a tensorial description, we start with the definition of displacements. We consider a point  $x$  of a body undergoing some deformation. After the deformation, the very same point has moved to a different location  $x'$ . The displacement  $u$  of point  $x$  is described by

$$u_i = x'_i - x_i. \quad (3.45)$$

Now, we consider a second point infinitesimally close to  $x$ . That point's coordinates are given by  $x + dx$ . Again, by deforming the body, we change the position of the two points, as illustrated in figure 3.3. The new positions are given by

$$x'_i = x_i + u_i(x), \quad (3.46)$$

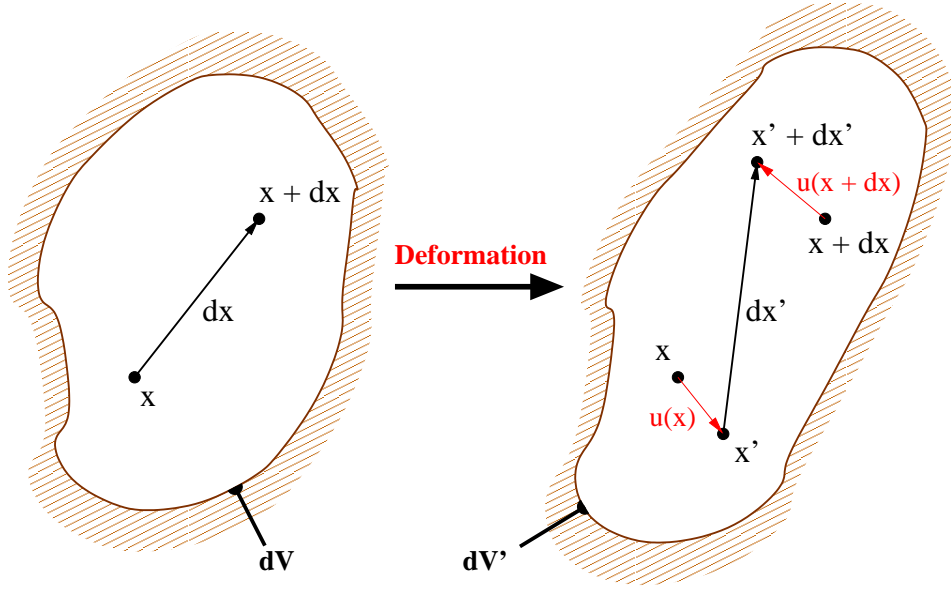


Figure 3.3: Deformation of an infinitesimal volume  $dV$ , containing the point  $x$  and its infinitesimal neighbor  $x + dx$ . After the deformation, the distance  $dx$  of the two points has changed to  $dx'$ , which is due to the change of deformation  $u(x)$  to  $u(x + dx)$ , when moving from  $x$  into the direction of  $dx$ .

and

$$\begin{aligned} x'_i + dx'_i &= x_i + dx_i + u_i(x + dx) \\ &= x_i + dx_i + u_i(x) + du_i. \end{aligned} \quad (3.47)$$

The function  $u(x)$  represents the displacement of point  $x$  (as in (3.45)), while  $u(x + dx)$  represents the displacement of point  $x + dx$ . The displacements are not equal in general, but differ by  $du$ .  $du$  can be seen as the change of displacement when moving from point  $x$  into the direction of  $dx$ . Thus, we can rewrite (3.47) as

$$x'_i + dx'_i = x_i + dx_i + u_i(x) + \frac{\partial u_i}{\partial x_j} dx_j. \quad (3.48)$$

The distance of the two points after the deformation is then given by

$$dx'_i = dx_i + \frac{\partial u_i}{\partial x_j} dx_j. \quad (3.49)$$

We can measure the distance of the two points by the Euclidean norm for vectors, denoted by  $\|\cdot\|$ . The distance of the points before the deformation reads

$$\|dx\| = \sqrt{dx_1^2 + dx_2^2 + dx_3^2}, \quad (3.50)$$

while the distance of the points after the deformation reads

$$\|dx'\| = \sqrt{dx_1'^2 + dx_2'^2 + dx_3'^2}. \quad (3.51)$$

The square of the distances is

$$\|dx\|^2 = dx_i dx_i \quad (3.52)$$

and

$$\begin{aligned} \|dx'\|^2 &= dx'_i dx'_i \\ &= \left( dx_i + \frac{\partial u_i}{\partial x_j} dx_j \right)^2 && \text{from (3.49)} \\ &= dx_i dx_i + 2 \frac{\partial u_i}{\partial x_j} dx_i dx_j + \frac{\partial u_i \partial u_i}{\partial x_j \partial x_k} dx_j dx_k. \end{aligned} \quad (3.53)$$

(The parts of (3.53) having several index pairs are expanded successively by the Einstein summation convention, i.e. the index pairs are replaced recursively, yielding multiple expansions.) We can further reorder (3.53) to

$$\|dx'\|^2 = \|dx\|^2 + 2\epsilon_{ij} dx_i dx_j, \quad (3.54)$$

where

$$\epsilon_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} + \frac{\partial u_k \partial u_k}{\partial x_i \partial x_j} \right). \quad (3.55)$$

Tensor  $\epsilon$  is called *strain tensor*. It describes the strain state at one point of a rigid body. Changes of the body's volume due to compression or elongation are represented by diagonal elements of  $\epsilon$ , while shear strains are represented by non-diagonal elements.

### 3.2.2 Stress Tensor

The deformation of a rigid body causes inner stresses within that body. We can describe these stresses again by a tensor, which is called *stress tensor*. By virtually cutting out a volume  $V$  of a deformed rigid body, we find that all outer forces acting on this volume are compensated by inner forces. The total inner force is given by the integral of the inner volume force  $f$  over volume  $V$

$$F_i = \int_V f_i dV. \quad (3.56)$$

We can write integral (3.56) as surface integral over the stress tensor  $\sigma$

$$F_i = \int_V f_i dV = \int_S \sigma_{ij} n_j dS, \quad (3.57)$$

where  $n$  is the outwards directed surface normal. By applying the Gaussian integration rule, we can transform the surface integral into a volume integral

$$F_i = \int_V f_i dV = \int_V \frac{\partial \sigma_{ij}}{\partial x_j} dV. \quad (3.58)$$

In differential form, we can write the equality of external force  $\widehat{F}$  and inner force as

$$\widehat{F}_i - \sigma_{ij} n_j dS = 0, \quad (3.59)$$

or, by using the volume integral of  $\sigma$  as

$$\widehat{F}_i - \frac{\partial \sigma_{ij}}{\partial x_j} = 0. \quad (3.60)$$

### 3.2.3 Kinetics and the Relation between Strain and Stress

The cause of motion of an elastic rigid body can be described by the equilibrium condition (3.60). In this context, we omit any volume forces such as gravity acting on the whole structure. By writing the external force  $\widehat{F}$  as change of momentum, we obtain the relation

$$\rho \frac{d^2 u_i}{dt^2} = \frac{\partial \sigma_{ij}}{\partial x_j}, \quad (3.61)$$

with  $\rho$  as density of the structure. In the Lagrangean view (cf. Section 3.1.2), the absolute derivatives can be directly replaced by partial derivatives

$$\rho \frac{\partial^2 u_i}{\partial t^2} = \frac{\partial \sigma_{ij}}{\partial x_j}. \quad (3.62)$$

In order to use equation (3.62) to compute displacements  $u$ , we have to express the stress tensor  $\sigma$  by the strain tensor  $\epsilon$  (cf. Section 3.2.1). This relation of stress to strain tensor describes the elastic behavior of the material considered. An approach for linear elastic materials is given by the *generalized Hooke's law*, which reads

$$\sigma_{ij} = C_{ijkl} \epsilon_{kl}. \quad (3.63)$$

The tensor  $C$  is called *elasticity tensor* and is a fourth order tensor. In three dimensional space, it consists of 81 elements. For isotropic materials, the elasticity tensor's amount of degrees of freedom can be reduced to only two, when symmetry properties and energy conservation laws are taken into account. Then, Hooke's law simplifies to

$$\sigma_{ij} = \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij}, \quad (3.64)$$

with  $\lambda$  and  $\mu$  being called *Lamé constants*.

Hooke's law describes only structures with linear elastic behavior. For the description of non-linear material behavior, there exists also a great variety of models, with the "Neo-Hook-materials" being one important class. However, we will not do any further investigations into these models here.

### 3.3 The Wet Surface - Coupling Equations

Up to now, we have considered the equations and laws describing fluids (cf. Section 3.1) and solids (cf. Section 3.2) separately in their respective fields of physics. The next step must be to understand how these two fields are coupled together or, in other words, how we can describe the interaction of these two fields by mathematical equations. Since we want to simulate a bidirectional fluid-structure interaction problem, we have to investigate both, the influence of the fluid on the structure and the influence of the structure on the fluid. The location of the interaction is limited to the common surface of fluid  $\Gamma_F$  and structure  $\Gamma_S$  - the wet surface - denoted by

$$\Gamma_{FS} = \Gamma_F \cap \Gamma_S. \quad (3.65)$$

Figure 3.4 shows an example geometry for a part of the common surface of a fluid and a structure.

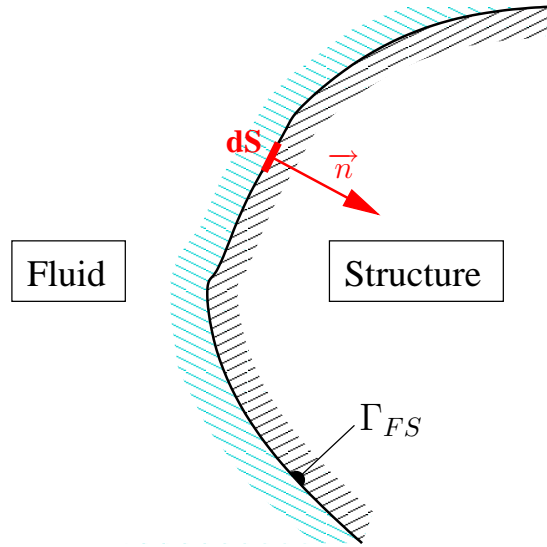


Figure 3.4: Fluid, structure and their common surface  $\Gamma_{FS}$ . The infinitesimally small area  $dS$  with normal vector  $\vec{n}$  is used to integrate the surface stresses.

We begin with the influence of the structure on the fluid. As a simplification, we limit ourselves to structures with smooth surface walls. Then, the usual assumption is, that the molecules (or fluid elements) of the fluid in direct contact with the surface of the structure are bound to it by atomic attraction forces. This implies the equality of fluid velocity  $v_F$  and change of solid displacements  $\partial u_S / \partial t$  at the surface

$$v_F = \frac{\partial u_S}{\partial t}, \quad \text{at } \Gamma_{FS}. \quad (3.66)$$

By this equation, the influence of the structure on the fluid is already completely described.

### 3 About Fluids and Structures

The influence of the fluid on the structure is given by the forces the fluid exhibits onto the surface of the structure. From Newton's third law, we can state an equilibrium of forces at the wet surface, reading

$$F_F + F_S = 0. \quad (3.67)$$

$F_F$  represents the force on the structure created by the fluid, while  $F_S$  represents the reaction force of the structure. Since these forces are acting on the surface  $\Gamma_{FS}$  only, we can rewrite (3.67) as surface integral of stresses

$$\int_{\Gamma_{FS}} \sigma_{ij}^F n_j dS + \int_{\Gamma_{FS}} \sigma_{ij}^S (-1) n_j dS = 0, \quad (3.68)$$

with  $n$  as normal vector of the fluid's boundary (i.e. outwards of the fluid domain, as shown in figure 3.4),  $\sigma^F$  the stress tensor of the fluid and  $\sigma^S$  the stress tensor of the solid. A factor of  $-1$  for the force of the structure must be added, since the direction of  $n$  is chosen to be normal to the fluid's surface. Finally, we can omit the integrals, by considering only an infinitesimally small part of the boundary  $\Gamma_{FS}$ , which gives us the second coupling condition, representing the influence of the fluid on the structure

$$\sigma_{ij}^F n_j dS = \sigma_{ij}^S n_j dS. \quad (3.69)$$

Summarizing the two coupling conditions at the common surface of fluid and structure, we have

$$\left. \begin{aligned} v_F &= \frac{\partial x_S}{\partial t} \\ \sigma_{ij}^F n_j dS &= \sigma_{ij}^S n_j dS \end{aligned} \right\} \text{ at } \Gamma_{FS}. \quad (3.70)$$

# 4 Simulation Tools and Employed Methods

This chapter discusses the simulation codes used as fluid and structure components of the coupled fluid-structure interaction simulation. These are F3F and AdhoC<sup>4</sup>. Both simulation tools have similar complexity and explanations of their features and theoretical background could fill a whole thesis on their own. The discussion in the following sections focusses on the description of F3F, since the structure solver was only used and, besides the integration of the new application programming interface described in Section 2.4, not enhanced or changed. The flow solver F3F, however, had to be partly restructured and enhanced to reach the full functionality required in an FSI simulation, described in Chapter 5. Nonetheless, after describing F3F in Section 4.1, also some explanations about the background of AdhoC<sup>4</sup> are following in Section 4.2.

## 4.1 F3F for Flow Simulations

As mentioned above, on the fluid side of the fluid-structure interaction problem, the software F3F was used in this thesis. F3F is a finite volume based solver for the incompressible Navier-Stokes equations, working with Cartesian grids. It has been developed by Maximilian Emans in [11] and extended with FSI capabilities (among many other features) by Markus Brenk in [4]. The following sections are mainly inspired by their work.

We will look at the advantages and disadvantages of Cartesian grids in Section 4.1.1, review the finite volume method for spatial discretization and see how it is used to discretize the Navier-Stokes equations in Section 4.1.2. The Chorin projection for discretizing the flow problem in time is discussed in Section 4.1.3 and, finally, we will see how forces on obstacles can be computed efficiently by the method of consistent forces in Section 4.1.4.

### 4.1.1 Cartesian Grids

F3F employs Cartesian grids in order to represent a given flow problem in space. Figure 4.1 shows an example for a simulation domain with a cylinder as obstacle discretized on a Cartesian grid. The grid is defined by equidistant orthogonal grid lines enclosing volumes called cells. A cell is marked as belonging to the cylinder, if it lies completely or partially within the cylinder's domain, as introduced in the marker-and-cell method [14]. Typical for Cartesian grids are the edgy surfaces originating in the orthogonal grid configuration. Of course, most of the geometries occurring in nature or technical applications do not have such a kind of surface, which appears to be a drawback of Cartesian grids. However, there exist methods to overcome this issue and gain further advantages in addition.

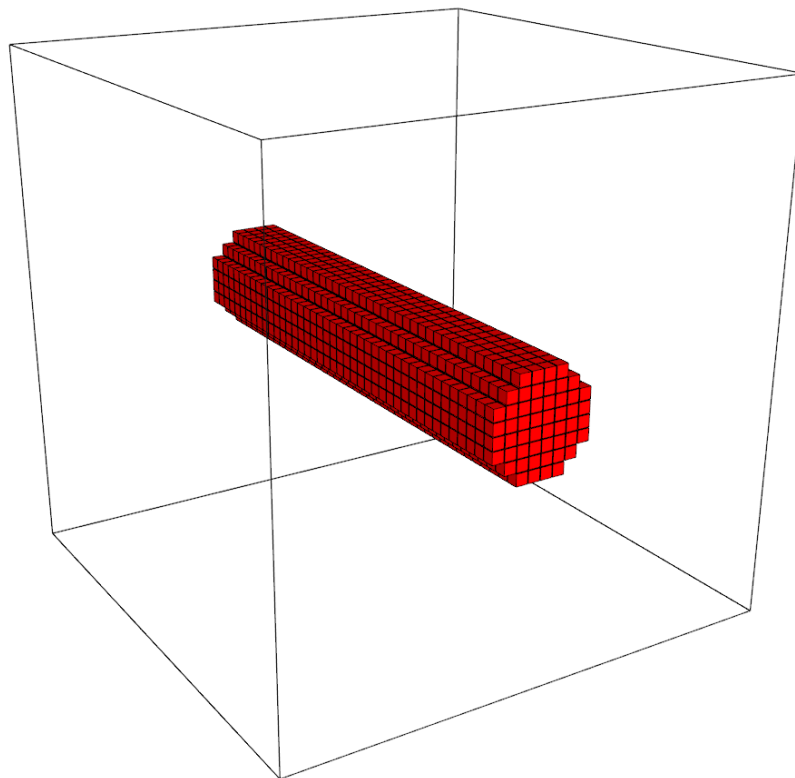


Figure 4.1: Example for a three dimensional fluid domain with a cylinder as obstacle discretized on a Cartesian grid, as employed by F3F.

Before coming to these methods, we will look at all the advantages gained by working on Cartesian grids. Cartesian grids belong to the class of structured grids, which need only minimal information to be described, compared to unstructured grids. Indeed, Cartesian grids are the most simple structured grids. An isovariate (i.e. same cellwidth in all spatial dimensions) Cartesian grid is fully described by its cellwidth  $h$ . The position

of any gridpoint  $x_{i,j,k}$  with indices  $i, j, k$  can be calculated via the formula

$$x_{i,j,k} = (ih, jh, kh).$$

This enables simple and efficient operations on the grid. As a consequence, the memory requirements are minimal, since the grid unknowns can be directly stored in array-like data structures without the need to store additional information such as relations between cells, faces, edges and grid points, as it appears for unstructured grids. Especially in fluid dynamics, one often needs a fine resolution in order to resolve all motion scales appearing in a flow and, thus, the need for an efficient storage scheme is high. The stream like storage format also enables a fast vectorized data processing employed by most modern computer hardware architectures.

To overcome the disadvantage of the rectangular surface approximation, several methods can be used. One class of methods is called ‘‘Cut Cell’’-methods [19]. It cuts the cells at the boundary of a geometry into non-orthogonal pieces, such that polygon-like approximations can be achieved. Another method is to choose the values of the boundary unknowns such, that their interpolated value at the position of the boundary matches the wanted boundary value [7]. The method of immersed boundaries provides one more technique to achieve a high approximation accuracy of boundaries [19]. In addition, surfaces of complicated geometries can be approximated with the help of local adaptive grid refinements [2]. A new flow solver supporting this feature and, at the same time, working with cache-efficient data structures and algorithms is currently under development.

### 4.1.2 Spatial Discretization by the Finite Volume Method

The finite volume method (FVM) (cf. Remark 4.1) is one common form of spatial discretization used for CFD problems. Other popular choices are the finite element method (FEM) (cf. Section 4.2) and the finite difference method (FDM).

F3F uses the finite volume method to discretize the Navier-Stokes equations derived in Chapter 3 in space. We recall the dimensionless form of the equations to be

$$\begin{aligned} \frac{\partial v_i}{\partial t} + v_j \frac{\partial v_i}{\partial x_j} &= -\frac{\partial p}{\partial x_i} + \frac{1}{Re} \frac{\partial^2 v_i}{\partial x_j \partial x_j} + \frac{1}{Fr^2} f_i, \\ \frac{\partial v_i}{\partial x_i} &= 0. \end{aligned}$$

Following the discretization methodology of the FVM, we have to integrate these equations over so-called control volumes  $V$ , which are in our case the cells of the Cartesian grid (cf. Section 4.1.1) used by F3F. Integrating the Navier-Stokes equations yields

$$\int_V \frac{\partial v_i}{\partial t} dV + \int_V v_j \frac{\partial v_i}{\partial x_j} dV = - \int_V \frac{\partial p}{\partial x_i} dV + \frac{1}{Re} \int_V \frac{\partial^2 v_i}{\partial x_j \partial x_j} dV + \frac{1}{Fr^2} \int_V f_i dV, \quad (4.1)$$

**Remark 4.1.** Finite volume method

The finite volume method is a method to discretize a continuous partial differential equation (PDE) in space. In the finite volume method the domain of interest  $\Omega$  is decomposed into a number of control volumes  $\Omega_i$  which fulfill the following properties:

- each  $\Omega_i$  is open, simply connected and has a polygonal boundary.
- $\Omega_i \cap \Omega_j = \emptyset$ , for  $i \neq j$ . This implies, that the control volumes are not overlapping.
- $\cup_{i=1}^M \overline{\Omega}_i = \overline{\Omega}$ , with overlines indicating the closure of  $\Omega$ , i.e. the set  $\Omega$  including its boundaries.

After decomposing the domain  $\Omega$  into control volumes  $\Omega_i$ , the PDE under consideration is integrated over each control volume and the Gaussian integration theorem is applied to transform the volume integrals into surface integrals. Then, there are two classes of finite volume methods, called cell-centered and node-centered approaches. The cell-centered approach assigns to each control volume a function value, such as the pressure of a fluid, while the node-centered approach assigns function values to the grid nodes, i.e. the intersection points of the grid lines. Figure 4.2 illustrates these two classes of approaches and shows a semi-staggered and fully-staggered approach in addition. The quadrature over the control volumes or volume surfaces respectively, via some numerical integration method finally leads to a system of equations, providing an approximate solution of the considered problem.

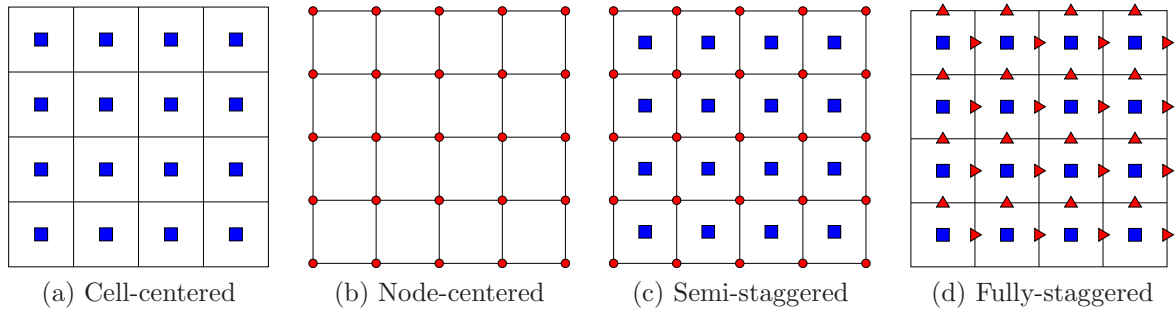


Figure 4.2: Basic classes of finite volume discretizations, with circles, squares and arrows denoting the unknowns in the grid. While in (a) and (b) the unknowns are located either in the cell center or at the cell corners, (c) combines these two forms and (d) further splits the unknowns of the cell corners into the cell edges. (c) and (d) are used for systems of equations with several unknown variables, of course.

and

$$\int_V \frac{\partial v_i}{\partial x_i} dV = 0. \quad (4.2)$$

Then, we transform the integrals of the convective, diffusive and pressure term in the momentum equations and the integral of the continuity equation into surface integrals and obtain

$$\int_V \frac{\partial v_i}{\partial t} dV + \int_S v_j v_i n_j dS = - \int_S p \delta_{ij} n_j dS + \frac{1}{Re} \int_S \frac{\partial v_i}{\partial x_j} n_j dS + \frac{1}{Fr^2} \int_V f_i dV, \quad (4.3)$$

and

$$\int_S v_i n_i dS = 0, \quad (4.4)$$

with  $n$  as the normal vector of the surface and  $\delta$  as unit tensor.

If we knew the velocity and pressure distribution of the fluid on the surface of each discretization cell, we could compute a solution of the integrated Navier-Stokes equations (4.3) and (4.4), valid for every control volume  $V$ . Since we do not have this information, we have to think about a proper restriction of velocity and pressure to a discrete, i.e. finite, set of unknowns and interpolation rules allowing it to compute approximations of the surface integrals. F3F uses a special choice for achieving this goal, described in detail in the work of Emans [11] and Brenk [4]. Figure 4.3 illustrates the positions of the unknowns in the semi-staggered discretization grid of F3F, by taking out one cell of the Cartesian grid.

Finally, by discretizing every spatial derivative of the Navier-Stokes equations (4.3) and (4.4), we arrive at the *semi-discrete Navier-Stokes equations*, reading

$$\mathbf{\Omega} \frac{\partial v_h}{\partial t} + \mathbf{C}(v_h) v_h = -\mathbf{M} p_h + \frac{1}{Re} \mathbf{D} v_h + f_h, \quad (4.5)$$

$$\mathbf{M}^T v_h = 0. \quad (4.6)$$

There, the index  $h$  annotated to the variables indicates the discretization. All variables are actually vectors, consisting of all unknowns of its type with a certain ordering. The variable  $v_h$  for example, consists of all velocity unknowns of the flow problem, i.e. all velocities in  $x, y$  and  $z$  direction. The unknown vectors are multiplied by discrete operators in the form of matrices, denoted by bold letters. The matrix  $\mathbf{\Omega}$  is called mass matrix. It represents the weights of the different cells on each other and simplifies to a diagonal matrix in the case of equidistant grids for our finite volume discretization.  $\mathbf{C}(v_h)$  describes the non-linear convective transport of velocity and is dependent on the velocity field  $v_h$  itself. The matrix  $\mathbf{M}$  forms the gradient of the pressure  $p_h$  and is the transposed of the matrix  $\mathbf{M}^T$  representing the discrete divergence operator.  $\mathbf{D}$  approximates the influence of molecular transport of momentum, namely diffusion and is scaled by the Reynolds number  $Re$ .

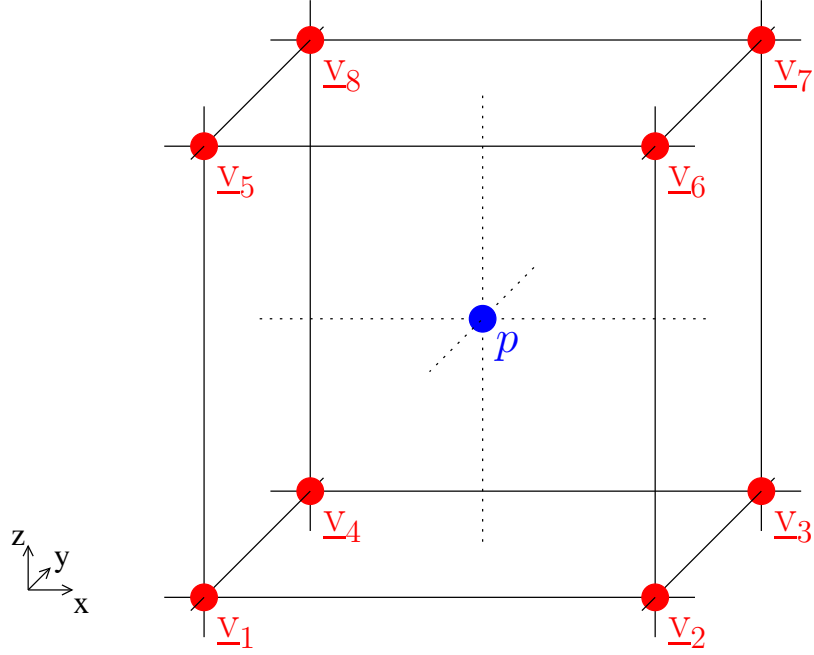


Figure 4.3: Semi-staggered finite volume discretization scheme as employed by F3F. Pressure unknowns  $p$  are located in the center of each cell, while velocity unknowns  $\underline{v}$  are located at the cell's corners. The numbering of the velocity vectors shown is the local, i.e. cell related, numbering. In addition, every unknown has also a global unique index.

### 4.1.3 Time Discretization by the Chorin Projection

The semi-discrete Navier-Stokes equations (4.5) and (4.6) (cf. Section 4.1.2) are only discretized in space. They still miss a discretization in time to yield a solvable discrete version of the fluid dynamics problem. There are several methods known for discretizing the Navier-Stokes equations in time, the one used by F3F is called the *Chorin projection* and has been introduced first by Chorin in [8]. It has the property of conserving zero divergency of the velocities and hence, conserve the overall mass of a fluid.

The starting point for deriving the Chorin projection is the semi-discrete form of the momentum equations (4.5), reading

$$\Omega \frac{\partial v_h}{\partial t} + \mathbf{C}(v_h)v_h = -\mathbf{M}p_h + \frac{1}{Re}\mathbf{D}v_h + f_h.$$

We reorder this equation to have the time derivative separated from all other terms

$$\frac{\partial v_h}{\partial t} = \Omega^{-1} \left( \frac{1}{Re}\mathbf{D}v_h - \mathbf{C}(v_h)v_h - \mathbf{M}p_h + f_h \right). \quad (4.7)$$

Then, we can discretize the time derivative with a numerical integration method for ODEs, where we will use the simple Euler method as described in Appendix A. Of

course, also schemes with higher approximation order can be applied here. We get

$$\frac{v_h^{n+1} - v_h^n}{dt} = \Omega^{-1} \left( \frac{1}{Re} \mathbf{D}v_h^n - \mathbf{C}(v_h^n)v_h^n - \mathbf{M}p_h + f_h^n \right), \quad (4.8)$$

where  $n$  and  $n + 1$  denote variables of the current and the next time step, respectively and  $dt$  is the length of one time step. The superscript  $n$  is intentionally omitted for the pressure term, we will see later why. By reordering (4.8) for the the velocity  $v_h^{n+1}$ , we obtain

$$v_h^{n+1} = v_h^n + dt\Omega^{-1} \left( \frac{1}{Re} \mathbf{D}v_h^n - \mathbf{C}(v_h^n)v_h^n - \mathbf{M}p_h + f_h^n \right), \quad (4.9)$$

We cannot yet solve equation (4.9), because we do not know where to take the pressure  $p_h$  from. To continue, we use the discrete continuity equation (4.6)

$$\mathbf{M}^T v_h = 0$$

as an additional condition for the velocity  $v_h^{n+1}$  as defined in (4.9). We obtain

$$\mathbf{M}^T v_h^n + dt\mathbf{M}^T \Omega^{-1} \left( \frac{1}{Re} \mathbf{D}v_h^n - \mathbf{C}(v_h^n)v_h^n - \mathbf{M}p_h + f_h^n \right) = 0. \quad (4.10)$$

Reordering (4.10) for the pressure  $p_h$  yields a *pressure Poisson equation* to be solved, reading

$$\mathbf{M}^T \Omega^{-1} \mathbf{M}p_h = -\frac{1}{dt} \mathbf{M}^T (v_h^n + dt\mathbf{M}^T \Omega^{-1} F^n), \quad (4.11)$$

with the *intermediate velocities*

$$F^n = \frac{1}{Re} \mathbf{D}v_h^n - \mathbf{C}(v_h^n)v_h^n + f_h^n. \quad (4.12)$$

Finally, after computing a solution for the pressure field  $p_h$ , we can compute the velocities of the next time step by

$$v_h^{n+1} = v_h^n + dt\Omega^{-1} (F^n - \mathbf{M}p_h). \quad (4.13)$$

Now, we also see why the pressure  $p_h$  does not have any superscript  $n$  or  $n + 1$ . The pressure computed does not belong to time  $t^n$ , but also not fully to time  $t^{n+1}$ , since the final velocities  $v_h^{t+1}$  are computed after solving the Poisson equation for the pressure term  $p_h$ .

Summarizing the Chorin-Projection, one can define three main steps:

1. Computation of intermediate velocities  $F^n$  from terms dependent on old velocities (eq. 4.12).
2. Set-up and solution of a pressure Poisson equation to obtain the pressure  $p_h$  (eq. 4.11)
3. Projection of the intermediate velocities into divergence-free velocities  $v_h^{n+1}$  by use of the computed pressure values (eq. 4.13)

#### 4.1.4 Computation of Forces on Obstacles

A fluid solver that is intended to be used in a coupled FSI simulation has to be able to compute forces caused by the fluid motion and exhibited onto obstacles in the fluid flow as described by the coupling condition (3.70) in Section 3.3. One possible approach for the computation of these forces is given by explicitly computing the stress tensor  $\sigma$  of the Navier-Stokes equations. However, F3F uses a different approach to compute forces on obstacles, which is called the *method of consistent forces* and has been introduced for finite volume methods in [13] and for F3F in [6].

As with the explicit computation of the surface stress tensor  $\sigma$ , the method of consistent forces uses the equilibrium of forces (3.70) at the common interface of fluid and structure  $\Gamma_{FS}$  and computes forces on this surface created by the fluid flow. However, it directly utilizes the surface stress tensor terms computed in turn of the Chorin projection, as introduced in Section 4.1.3.

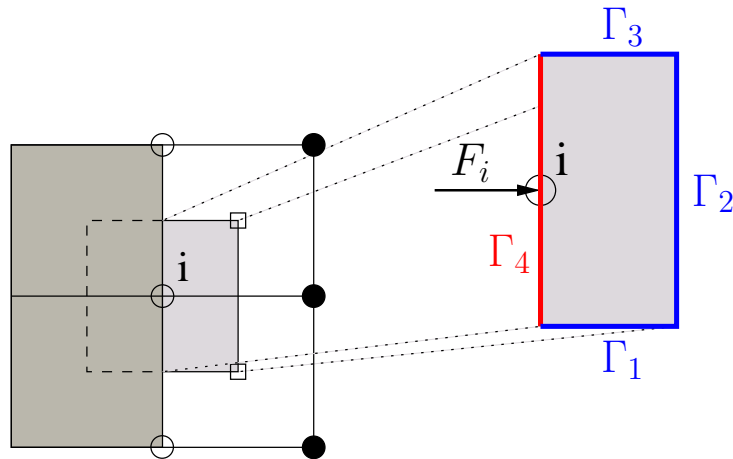


Figure 4.4: Part of the fluid-structure surface  $\Gamma_{FS}$  in a two-dimensional Cartesian grid of a fluid solver, with magnified control volume of the momentum equations.

In order to compute the forces on the surface of the structure, we have to think in terms of control volumes. Consider a two-dimensional version of a Cartesian grid as shown at the left side of Figure 4.4. The two left cells in dark gray are solid cells, belonging to a structure, while the two right cells are fluid cells. The circles denote velocity unknowns, where filled circles indicate real degrees of freedom and the empty circles at the boundary of the structure are boundary values for the fluid. The small squares denote pressure unknowns of the fluid. Positioned in the middle of the four cells is a node with index  $i$ , that has a control volume (related to the momentum equations) indicated by a second rectangle, lying half in the structure and half in the fluid domain. The part of this control volume belonging to the fluid domain is zoomed to the right side of the figure. In the finite volume method (cf. Section 4.1.2), the forces on the faces  $\Gamma_1$ ,  $\Gamma_2$  and  $\Gamma_3$  are computed in turn of solving the Navier-Stokes equations. The sum

of these three forces must be in equilibrium with the reaction force  $F_i$  of the structure, which is the integral of the stress tensor  $\sigma_i^S$  from (3.70) over surface  $\Gamma_4 \subset \Gamma_{FS}$ .

These considerations lead to the same results for corner scenarios and three-dimensional scenarios in an analogous manner, which is a semi-discrete formulation for the equilibrium of the forces acting on the control volume

$$F_i = -(\Omega \frac{\partial v_h}{\partial t} + \frac{1}{Re} \mathbf{D}v_h + \mathbf{C}(v_h)v_h - \mathbf{M}p_h + f_h)_i. \quad (4.14)$$

For a non-accelerated structure surface, the term  $\Omega \partial v_h / \partial t$  equals zero. Note that the computation of the terms on the right side of (4.14) can be achieved nearly for free, since these terms have to be computed already for the Chorin projection and can be reused.

The disadvantage of the method of consistent forces is, that the single force contributions of convective, diffusive and pressure terms can not be separated, only the total sum of forces is obtained. The advantages are a computation of forces on structure surfaces for nearly no extra costs and the preservation of the accuracy of the consistent forces in the order of the primary variables  $v$  and  $p$ .

## 4.2 AdhoC for Structure Simulations

To simulate structures, the simulation program AdhoC<sup>4</sup> (henceforth called AdhoC) was used in this thesis. AdhoC has been developed at the “Chair for Computation in Engineering“ at the TU München and is an hp-adaptive finite element solver, i.e. a finite element solver employing mesh refinements and high-order ansatz functions to get a solution of high accuracy. The following two sections give an introduction to the finite element method and to high order ansatz functions. They are mainly taken from [20], a more detailed description of the  $hp$ -method can be found in [10].

### 4.2.1 The Finite Element Method for Structure Computations

The finite element method (FEM) is based on the Galerkin method for finding approximate solutions to partial differential equations (PDEs) defined on a domain  $\Omega$  with given boundary values. For any PDE, we can bring all its terms on one side of the equation, which yields the *residual form*

$$R(x, u) = 0 \quad \text{in } \Omega. \quad (4.15)$$

The solution  $u$  of this equation is considered to be a *strong solution*, since it is must be valid for every point  $x \in \Omega$ . We can relax this condition and search for a *weak solution*

$u_w$  of the boundary problem, which has to fulfill the PDE in the integral form multiplied by an arbitrary *test function*  $v$

$$\int_{\Omega} R(x, u_w) v dx = 0. \quad (4.16)$$

Moreover, we search for a solution  $u_w$  in a so-called *Hilbert space*  $H$  only, because it can be shown that a solution for the *weak form* (4.16) exists in this space and is unique. Since a solution in the infinite dimensional space  $H$  is impossible to find by numerical methods, we restrict ourselves again and search for a solution in a suitable finite dimensional subspace  $\tilde{H} \subset H$ , whose basis functions  $\phi_i \in \tilde{H}, i = 1, \dots, n$  we know. At the same time, we restrict the test function space to the same subspace, such that  $u_w, v \in \tilde{H}$ . Then, the functions  $u_w$  and  $v$  can be expressed by a linear combination of basis functions, reading

$$u_w = \sum_{i=1}^n u_i \phi_i \quad (4.17)$$

$$v = \sum_{i=1}^n v_i \phi_i. \quad (4.18)$$

By inserting (4.17) and (4.18) into (4.16), we can write it as

$$\sum_{i=1}^n v_i \int_{\Omega} \left( R(x, \sum_{j=1}^n u_j \phi_j) \phi_i dx \right) = 0 \quad (4.19)$$

where the  $v_i$  are arbitrary, since we chose  $v$  to be arbitrary. Because (4.19) is valid for any arbitrary  $v$  and since we can write every  $v$  as a linear combination of its basis functions, it suffices to test equation (4.19) with the basis functions  $\phi_i$  only. Then, we can write our PDE in weak form as

$$\int_{\Omega} R(x, \sum_{j=1}^n u_j \phi_j) \phi_i dx = 0 \quad \text{for } i = 1, \dots, n. \quad (4.20)$$

The further procedure of the finite element method is to divide the considered domain  $\Omega$  into subdomains  $\Omega_i$  such that

$$\bigcup_i \Omega_i = \Omega. \quad (4.21)$$

The  $\Omega_i$  are called cells and its vertices *mesh nodes*. In addition, we consider only basis functions  $N_k$  with compact support, i.e. every function  $N_k$  is non-zero only on a part of  $\Omega$ . We assign one basis function to every mesh node and define their support to be equal to the adjacent finite elements. The  $N_k$  are called *shape functions* in finite element terminology.

By applying the decomposition into cells and the special choice of shape functions  $N_k$  to the Galerkin method, we can write our PDE in weak form as

$$\sum_k \int_{\Omega} R(x, \sum_{j=1}^n u_{kj} N_{kj}) N_{ki} dx = 0 \quad \text{for } i = 1, \dots, n. \quad (4.22)$$

This form allows a divide & conquer scheme to solve a given partial differential equation, which is well suited for a solution on a computer. By applying some, often numerical, integration to every cell, a solution to (4.22) can be found.

### 4.2.2 High Order Shape Functions

There exist several possibilities to increase the accuracy of a solution obtained by the finite element method. Probably the most common method is called  $h$ -adaptivity, and means a refinement of the finite element mesh. The convergence rate achieved by this approach is  $O(h)$ , with  $h$  representing the meshwidth, i.e. the maximal distance of two adjacent mesh nodes. A different approach on getting higher accuracy is the  $p$ -adaptivity. The order of the shape functions used to solve a PDE with the FEM is usually denoted by a  $p$ . Thus, in the  $p$ -adaptive approach, one increases the order of the shape functions, which leads to an exponential increase of accuracy for smooth problems.

The most common choice for high order shape functions are the orthogonal set of Legendre polynomials  $L_n(x)$ . These are defined as

$$L_n(x) = \frac{(-1)^n}{2^n n!} \frac{d^n}{dx^n} [(1-x)^n (1+x)^n]. \quad (4.23)$$

AdhoC uses the following set of one-dimensional shape functions

$$\begin{aligned} N_1(\xi) &= \frac{1}{2}(1 - \xi) \\ N_2(\xi) &= \frac{1}{2}(1 + \xi) \\ N_i(\xi) &= \phi_{i-1}(\xi) \quad i = 3, 4, \dots, p + 1, \end{aligned}$$

with

$$\begin{aligned} \phi_j(\xi) &= \sqrt{\frac{2j-1}{2}} \int_{-1}^{\xi} L_{j-1}(x) dx \\ &= \frac{1}{\sqrt{4j-2}} (L_j(\xi) - L_{j-2}(\xi)) \quad \text{for } j = 2, 3, \dots \end{aligned}$$

and  $\xi$  as local coordinate  $\Omega = \{|\xi| - 1 \leq \xi \leq 1\}$ . The first five shape functions are illustrated in Figure 4.5. Shape functions for higher dimensions can be obtained by forming a tensor product for every dimension, where this approach allows to have shape functions with different order  $p$  for each dimension.

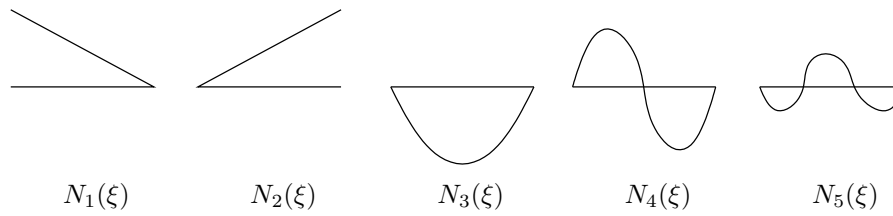


Figure 4.5: 1d shape functions as employed by AdhoC for  $p$ -adaptive accuracy control.

# 5 F3F - Preparing a Fluid Solver for FSI Simulations

After introducing the main concepts of the fluid simulation program F3F in Chapter 4, we will look at the extensions and modifications done in this work, in order to prepare F3F for the computation of the fluid-structure interaction benchmarks presented in Chapter 6. In Section 5.1 the implementation of new boundary types is discussed. Section 5.2.2 describes another important task, the re-implementation of the FSI functionalities of F3F. The last section in this chapter, Section 5.3, gives a short overview on three more tasks done during this thesis.

## 5.1 Extensions of Boundary Types

The fluid simulation program F3F is restricted to simulate three-dimensional scenarios only. In order to compute results comparable to the two-dimensional (2d) FSI benchmark scenario introduced in Chapter 6, a special three-dimensional (3d) scenario had to be set up. This scenario, henceforth called quasi-2d scenario, has to offer the possibility to easily rescale the results obtained from it into 2d equivalent results. This is achieved by a special choice of boundary conditions, consisting of no-slip, free-slip, in- and outflow boundaries. Since not all of these boundary types were implemented in F3F, extensions had to be done. We will look at the boundary types used for the simulation of the FSI benchmark in Section 5.1.1. Section 5.1.2 discusses the exact setup of these boundary conditions to achieve a quasi-2d scenario for the FSI benchmark.

### 5.1.1 Fundamental Boundary Types

In a general sense, boundary conditions for boundary value problems can be divided into *Dirichlet* and *Neumann boundary conditions* (cf. Remark 5.1). We will see how the boundary types used in fluid dynamics can be categorized into these two classes and mixed forms. In the following paragraphs, the boundary types used for the benchmark scenario are introduced. We refer to the boundary of the fluid domain as  $\Gamma$ .

**Remark 5.1.** Dirichlet and Neumann boundary conditions

Boundary conditions for boundary value problems (BVPs) can be divided into the two classes Dirichlet and Neumann boundary conditions. Both boundary conditions prescribe some conditions for the primary variables  $\psi$  at the boundary  $\Gamma$  of the considered BVP. For the Dirichlet boundary condition, this condition reads

$$\psi = \alpha_D \quad \text{at } \Gamma,$$

with  $\alpha_D$  as prescribed solution of the BVP at its boundary. The Neumann boundary condition prescribes the solution for the (partial) derivative of the primary variable with respect to the dimensions of the BVP, reading

$$\frac{\partial \psi}{\partial x} = \alpha_N \quad \text{at } \Gamma.$$

**No-slip boundary** The no-slip boundary uses the classical Dirichlet boundary condition for the velocity  $v$  of the fluid flow. Fluid elements in direct contact to a no-slip boundary are considered to be fixed to it, i.e. their velocity has to be equal to that of the no-slip boundary, reading

$$v_i = v_i^D \quad \text{at } \Gamma. \quad (5.1)$$

No-slip boundaries can be used to model the boundary walls of a fluid domain or a static obstacle, by prescribing a zero velocity

$$v_i = 0 \quad \text{at } \Gamma \quad (5.2)$$

in all directions of the considered problem. This hinders the fluid to flow through the wall, but also to flow in wall-tangential direction (when directly at the wall). No slip boundaries are also used to model dynamic obstacles for FSI simulations. Here the prescribed velocity is in general different from zero (as for inflow boundaries), modeling the velocity of the obstacle.

**Inflow or inlet boundary** Inflow boundaries are also Dirichlet boundaries, usually prescribing a certain velocity profile in wall normal direction. They are given by

$$v_i = v_i^{in}(x, t) \quad \text{at } \Gamma. \quad (5.3)$$

As indicated, the prescribed velocity may depend on the location  $x$ , when applying a parabolic inflow profile, for example, but can also depend on time.

**Outflow or no-force boundary** To model the outflow of fluid from the flow domain, so-called outflow boundaries are used. Since one does not know the state of the fluid

behind an outflow boundary, one makes the assumption that the flow field after the outflow is similar to the flow inside the domain simulated. This implies that the fluid field is not changing at the outflow, which leads to a Neumann condition, reading

$$\frac{\partial v_i}{\partial x_j} = 0 \quad \text{at } \Gamma. \quad (5.4)$$

**Free-slip boundary** A free-slip or slipwall boundary is basically an artificial boundary type, since it assumes zero friction in boundary tangential direction. Thus, this boundary cannot be used to model physical walls but for inner boundaries such as symmetry planes. This boundary is one essential ingredient used to model the quasi-2d benchmark scenario. The mathematical description of the boundary condition reads

$$v_i n_i = 0 \quad \text{at } \Gamma, \quad (5.5)$$

with  $n$  as wall normal vector. For the wall tangential directions, no conditions are prescribed, such that the fluid can move freely in this directions.

### 5.1.2 A Quasi-2D Scenario in Three Dimensions

We will look at the quasi-2d setup for the FSI benchmark here. For a detailed description of the 2d benchmark, cf. Section 6.3. The following boundary types, as described in Section 5.1.1, are used in the quasi-2d scenario:

- inflow boundary,
- outflow boundary,
- no-slip boundary,
- free-slip boundary.

While inflow and no-slip boundaries had been already implemented in F3F, there were no outflow<sup>1</sup> and free-slip boundaries available and had to be implemented during this thesis. Test results of the new boundary types can be found in Chapter 6.

The discretization of the flow unknowns in F3F is based on a semi-staggered scheme (see Figure 4.2 (c) on page 45). Conforming to this scheme, the conditions prescribed at a boundary of the fluid domain lead to a prescription for every velocity node on the boundary. Since the boundary faces of the fluid domain share their edges with other

---

<sup>1</sup>F3F used Dirichlet boundaries to model the outflow of the fluid domain. Velocities equal to the ones at the inflow were prescribed. While this can lead to reasonable results for steady flow cases, it gets problematic in the case of the FSI benchmark. There, the velocities of the dynamic obstacle are constantly changing, which makes special correction schemes necessary to ensure the zero divergence of the flow.

boundary faces, overlaps of boundary conditions can occur at edges and corners of the domain. To solve this problem in case of differing boundary conditions, one of the boundary types of the faces has to be chosen for the respective edge. The exact setup of the quasi-2d scenario is illustrated in Figure 5.2, where the fluid domain is unfolded for an easier understanding. When viewing the domain from boundary 3, the setup of the 2d benchmark is obtained. Now, this 2d view is extended into a third dimension, the x-dimension. What we want to achieve, is to have the flow field constant in x direction, i.e. the influence of the flow in x-direction has to be minimized. The first necessary condition to achieve a constant flow field in x-direction, is a constant geometry in this direction. Hence, the 2d geometry is identically prolonged over the whole domain width, from boundary to boundary. The second condition to be fulfilled is a constant inflow profile in x-direction. In the benchmark scenario, a parabolic inflow profile is used. Hence, we apply this profile for every discretized layer of nodes along the x-direction of the domain inflow. Figure 5.1 illustrates the 3d inflow profile used for the FSI benchmark. Not to change this profile by the influence of friction at the boundaries 3 and 6, we use free-slip boundary conditions there. Applying all these conditions yields a constant flow profile in x-direction including the boundary layer of velocity nodes.

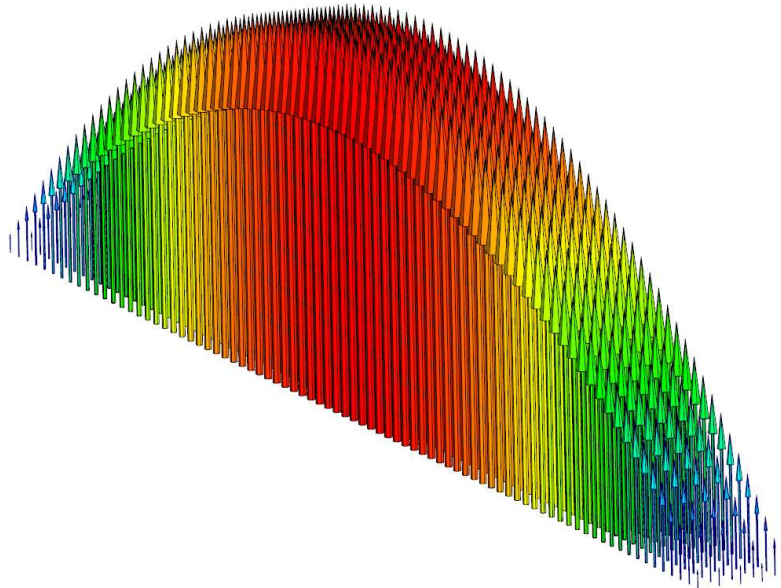


Figure 5.1: Inflow velocity profile applied for the simulation of the FSI benchmark scenarios with F3F.

The forces acting on the obstacle in the quasi-2d scenario correspond to the integral of the stress tensor over the surface of the obstacle. The reference values of the FSI benchmark are obtained in the same way, but the integration is done over a 2d surface only. However, this 2d surface integration is equivalent to an integration over a 3d geometry that has a width of  $w_{2d} = 1m$ . In order to scale the forces obtained in the

quasi-2d scenario of F3F, we need to compute a scaling factor  $s$  given by

$$s = \frac{w_{2d}}{w_{3d}}, \quad (5.6)$$

with  $w_{3d}$  as domain width in the third dimension. Then we can scale the forces obtained by

$$F_{2d} = sF_{3d} \quad (5.7)$$

and obtain forces comparable to the 2d reference forces of the FSI benchmark. Other quantities, such as pressure or the position of points on the obstacle changing with time can be compared directly.

## 5.2 Re-Implementation of FSI Functionality

In [9], the functionality of F3F has been extended to prepare it for coupled FSI simulations with FSI\*ce and a coupling supervisor with GUI viewer was introduced in order to visualize the coupling mesh data and the fluid solver state. All this functionality has been built around a new class of a Cartesian grid, called `CFDGrid`. During this thesis, the decision to remove the `CFDGrid` was made and the FSI functionality in F3F was re-implemented, exclusively based on F3F internal functionalities. In the following sections, the motivation to take this step is discussed, the required FSI functionalities are described and, finally, one completely new implemented feature, responsible for the handling of multiple obstacles, is shown.

### 5.2.1 Motivation for the Re-Implementation

The `CFDGrid` adds a significant amount of complexity to the handling of obstacles used for coupled FSI simulations, since it introduces an additional grid with its own coding interface and makes conversions between the Cartesian grid of F3F and the `CFDGrid` necessary. At the same time, the internal handling of dynamic, i.e. moving obstacles in F3F is well implemented and allows to implement the same functionalities as implemented with the help of `CFDGrid`, without introducing significant problems or increasing the implementation effort a lot.

Moreover, the `CFDGrid` needs a comparable amount of memory as the F3F internal grid, which halves the maximal problem size possible to be simulated. This issue is especially problematic for computational fluid dynamics, since in this simulation field mostly fine grid resolutions and, hence, high amounts of memory is required. The whole algorithmic design of F3F is indeed tailored to consume only few memory, which is another strong argument against the maintenance of the `CFDGrid`.

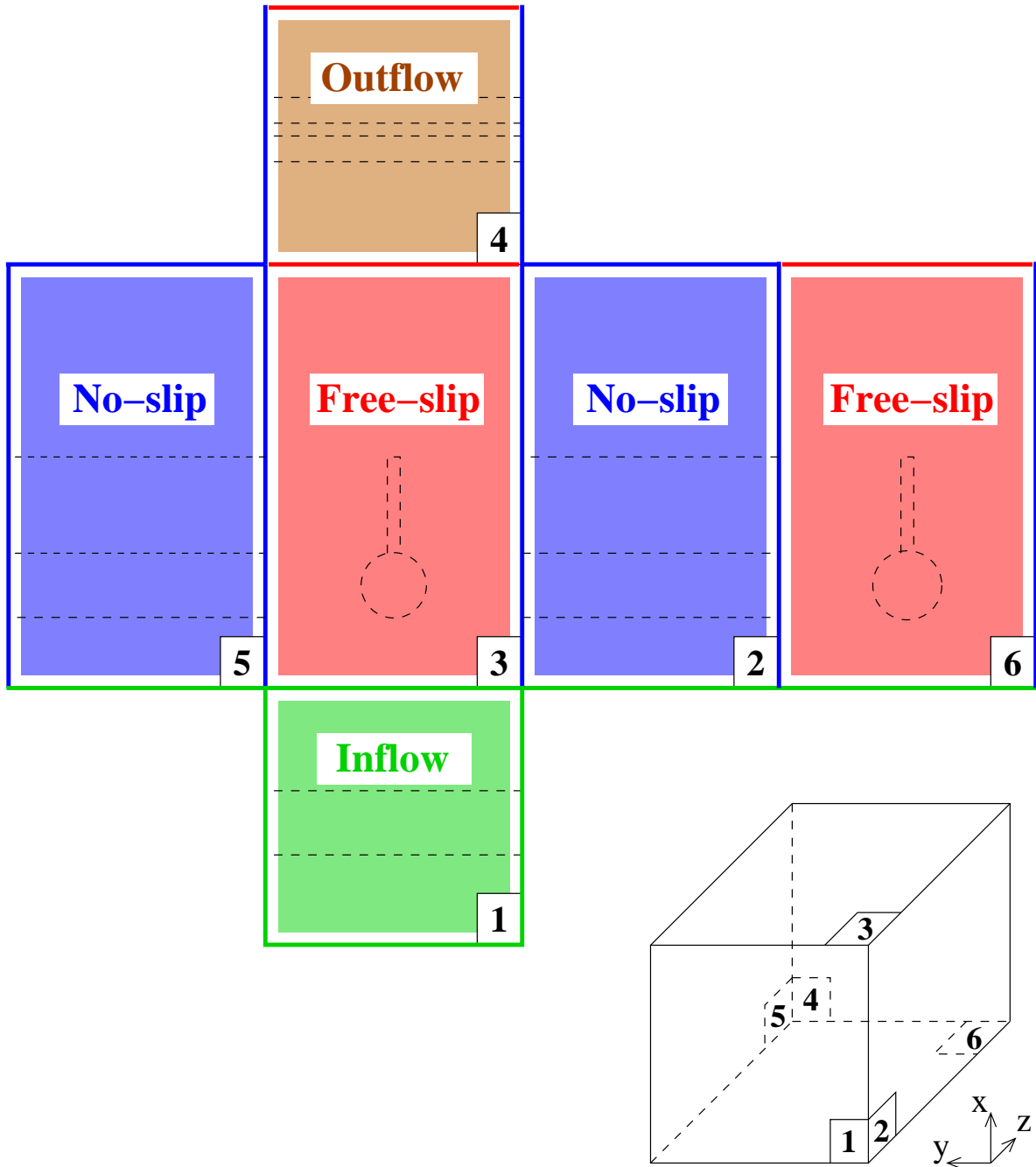


Figure 5.2: Unfolded boundaries of the 3d scenario used to compute the FSI benchmark with F3F. A 3d miniature version of the domain is shown on the bottom right of the figure. Drawn with dashed lines is the projected geometry of the obstacle, consisting of a cylinder with attached cantilever. Each side of the domain has assigned one boundary type. In addition, every edge (and also corner) of the domain has a defined boundary type, indicated by corresponding colors.

## 5.2.2 Re-Implemented Functionality

The functionalities that were re-implemented for F3F, in order to perform coupled FSI simulations with FSI\*ce, are enumerated here:

1. Conversion of the geometry between the triangulated coupling mesh and the Cartesian fluid solver mesh.
2. Reading of displacement values from the coupling mesh and mapping onto the fluid solver mesh.
3. Computation of forces on obstacles and conversion of these forces to stresses.
4. Handling of dynamic obstacle geometries, changing during a simulation run.
5. Handling of several obstacle geometries that may have common boundaries.

Point 5 is actually not a hard requirement for FSI simulations as long as only one obstacle is used in the FSI simulation. However, for the setup of the FSI benchmark scenario, two obstacles are used, a static cylinder and a cantilever attached to it. At the connection of the two obstacles, several problems occurred, which led to the development of a more sophisticated treatment for multiple obstacles, described in the next section.

## 5.2.3 Handling of Multiple Obstacles

F3F uses Cartesian grids to discretize spatial geometries (cf. Section 4.1.1) and further applies the finite volume method (cf. Section 4.1.2) to discretize the Navier-Stokes equations on these grids, using a semi-staggered approach. This choice of the discretization scheme leads to the consideration of cells and nodes in the domain of F3F, where each cell and node can be of a special type. Figure 5.3 illustrates the three types of cells and two types of (domain internal) nodes occurring in F3F. No equation is solved within solid cells, since all their unknowns are lying within the obstacle geometry and are, hence, not described by the Navier-Stokes equations. This is different for boundary cells having some nodes belonging to an obstacle and some belonging to the fluid domain. The Navier-Stokes equations are solved for the nodes belonging to the fluid domain, while the solid nodes are handled as no-slip Dirichlet boundary nodes, as described in Section 5.1.1. In addition, forces are computed at these boundary nodes by using the method of consistent forces described in Section 4.1.4.

For the case of an obstacle that is allowed to change its position during a simulation, as it will typically appear in FSI simulations, the setup of fluid, boundary and solid cells must be handled in a flexible way. The scheme applied to update the cell- and node types of F3F after an obstacle movement or a deformation of its geometry is enumerated here.

1. Removing all previous boundary cells belonging to the obstacle.

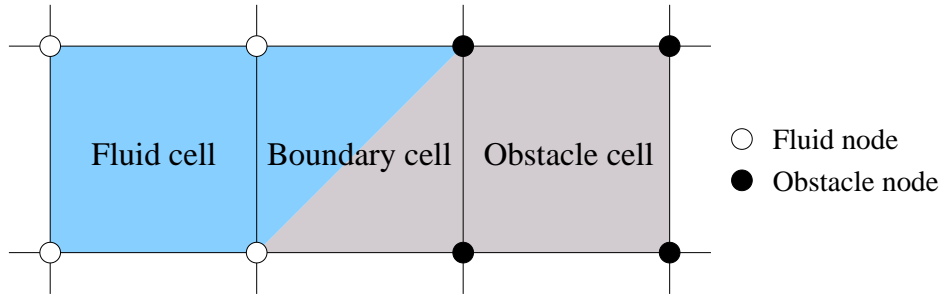


Figure 5.3: Inner-domain cell- and nodetypes used in F3F. The left cell is a fluid cell, consisting of fluid nodes only. The middle cell is a boundary cell, since it contains fluid and obstacle nodes. The right cell contains obstacle nodes only and is, hence, called obstacle cell.

2. Removing all previous solid cells and in the same turn converting all nodes belonging to these obstacle cells into fluid nodes.
3. Setting new obstacle cells belonging to the obstacle after the movement and converting all nodes of these cells to obstacle nodes.
4. Setting new boundary cells, originating from the new setup of obstacle cells and fluid cells.

When two or more obstacles are considered, a situation can occur, where the obstacles touch each other, i.e. a part of their outermost layer of obstacle nodes is coinciding, without any boundary cells in-between. This situation is illustrated in Figure 5.4. Now, if one of the obstacles changes its position such that it leaves the contact to the other obstacle, problems are occurring when using the updating scheme discussed: For example, only the obstacle with changed position is updated and in particular enhanced with new boundary cells, which could leave the remaining obstacle without any boundary cell layer and lead to a wrong setup in the system of equations to be solved. Besides this example, there are also other problems occurring which are not discussed here.

The basic idea to solve this problems is to introduce a memory for each cell, such that it knows to how many obstacles it belongs. This memory is implemented as a dedicated counter of how many obstacles are using the cell as boundary cell and another counter for how many obstacles are using it as obstacle cell. Figure 5.5 illustrates how the before described problem of the touching obstacles is solved by so-called “owner counters” of the cells.

### 5.3 Further Work Done

During this thesis, a lot of smaller tasks had to be done in order to prepare F3F for the computation of coupled FSI scenarios with FSI\*ce. In Sections 5.1 and 5.2 we already

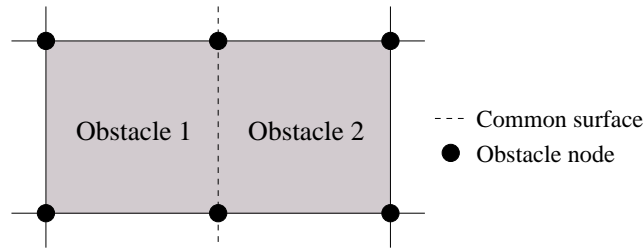


Figure 5.4: Touching obstacles in F3F. When several obstacles are discretized in the domain of F3F, they are allowed to touch each other. Then, a common surface exists where one layer of obstacle nodes is shared by the two obstacles and no layer of boundary cells is in-between.

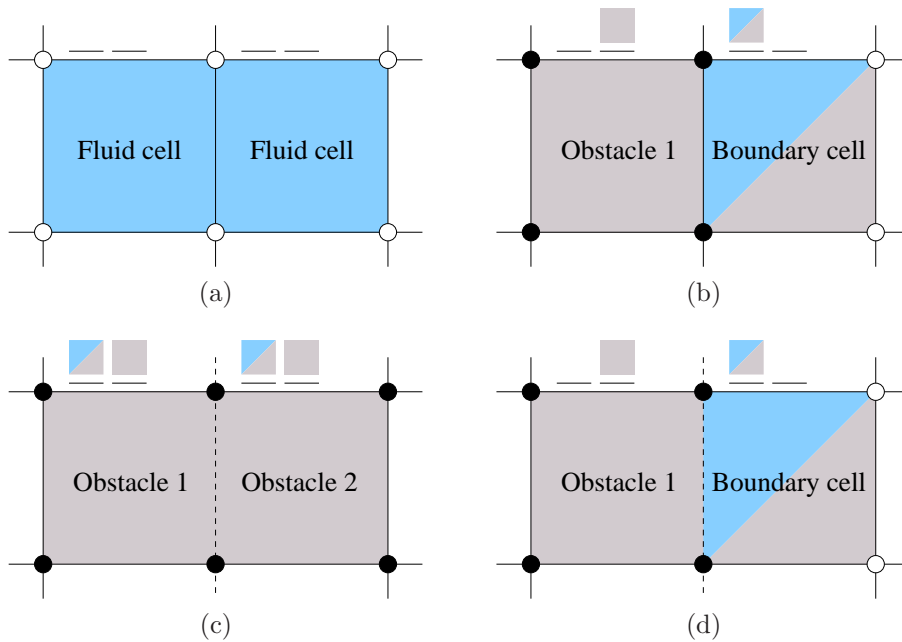


Figure 5.5: Example for handling multiple, moving obstacles with the help of cell owner counters. In (a), the two cells considered are both fluid cells, their owner counters are zero. Then, in (b) the left cell is converted to an obstacle cell, belonging to obstacle 1. The right cell is converted to a boundary cell, accordingly. The obstacle owner counter of the left cell is increased by one, illustrated by a small obstacle cell on top of the cell. For the right cell, the same happens for its boundary owner counter. A second obstacle is added in (c), touching obstacle 1. This increases the obstacle owners count of the right cell by 1, while its boundary owners count remains at 1. In addition, the boundary owners count of the left cell increases, despite the cell is not converted to a boundary cell. In (d), obstacle 2 has changed its geometry, such that the right cell does not any longer belong to it. This decrements the obstacle owner counter of the right cell, but does not convert it to a fluid cell, because its boundary owners counter is bigger than zero. Also, the boundary owners counter of the left cell is decremented, but the cell remains an obstacle cell, since its obstacle owner counter is bigger than zero.

discussed some important modifications and extensions made. This section gives an overview on three further tasks done during this thesis.

### 5.3.1 Internal FSI and Implicit Coupling

The fluid simulation program F3F also comprehends an internal mode for FSI simulations, used in [5], for example. To realize internal FSI simulations, F3F models structures as stiff objects undergoing translational and rotational acceleration forces. The forces are computed by the method of consistent forces, as introduced in Section 4.1.4.

One of the first tasks during this thesis was to use the internal FSI capabilities of F3F to compute some tests and implement a prototype for a coupling unit later implemented in FSI\*ce (cf. Chapter 2). The implemented class for implicit coupling is very similar to the implicit coupling functionality of the coupling unit described in Section 2.3.2. Some tests were performed to compare the influence of the implicit coupling scheme with the standard explicit scheme. The results can be found in Chapter 6.

### 5.3.2 Exchanging the Convective and Diffusive Operators

During the tests on the newly implemented boundary types, described in Section 5.1, problems occurred when increasing the Reynolds-number of the simulated flows. After some period of analysis, the cause of these effects turned out to be the discretized convective operator<sup>2</sup> of the Navier-Stokes equations. It was decided to replace the convective and the diffusive operators of F3F by the operators of the simulation tool PEANO. This is possible because the resulting discretization of the operators in PEANO is almost identical to that in F3F, with PEANO using a finite element approach to reach the discretized form of the Navier-Stokes equations. In addition, the exchange of operators makes only sense, because PEANO uses operators with different properties than those in F3F.

The main challenge when embedding the PEANO operators into F3F was a different enumeration scheme of the velocity unknowns. Conversion functions had to be written, to map unknowns from the F3F enumeration system to the PEANO enumeration system and vice versa. The PEANO operators were integrated as an option into F3F, making it possible to easily switch back to the native operators of F3F (cf. Appendix C). After the integration of the operators, tests were done for an empty channel. The results can be found in Chapter 6.

---

<sup>2</sup>Actually, we found no evident proof for the convective operator as cause of the problems. However, after a non-successful period of searching for failures in the source code of F3F and some review of the works of Emans and Brenk, we got convinced that there was a principal problem in the discretization scheme of the convective operator, appearing only at boundaries with Neumann conditions.

### 5.3.3 Integrating the New Coupling API

In Chapter 2, the extension of the coupling tool FSI\*ce by a more powerful coupling supervisor was described. To enable simulation programs to utilize the features of this new coupling supervisor, the application programming interface (API) of the solver library of FSI\*ce has been adapted. The next step to be done was to integrate the new API into the solvers to be coupled, which are F3F and the structure simulation program AdhoC.

The integration was done according to the description given in Section 2.4 following Listing 2.4 for both F3F and AdhoC. The amount of work to integrate the new API was very low and activated the support for all coupling schemes implemented in the coupling supervisor. Some tests were performed, the results can be viewed in Chapter 6.

# 6 Numerical Results

This chapter presents the numerical results obtained during this thesis. A glance on implicit time coupling methods is given in Section 6.1. There, the results of an internal FSI simulation with F3F are shown, comparing implicit and explicit time coupling methods. The following Section 6.2 shows the results for new boundary types applied to an empty channel flow. Finally, in Section 6.3, the 2d setup of the FSI benchmark scenario is given in detail and first results obtained are shown.

## 6.1 Internal FSI Simulations

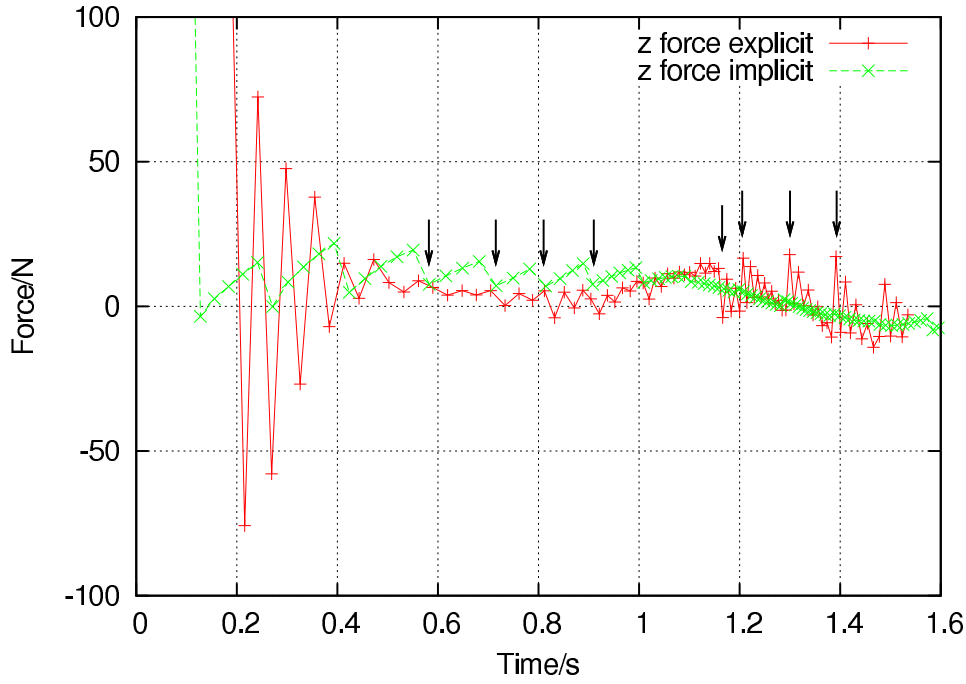
As described in Section 5.3.1, an implicit coupling scheme has been implemented for the internal FSI mode of F3F. Some comparisons were performed with the explicit coupling scheme by using a channel flow with inserted spherical obstacle. The results in Figure 6.1 compare forces and velocities in streamwise direction of the two coupling methods. Both methods show jumps every 3 to 5 time steps indicated by arrows in the figure. This is due to the discrete movement of the obstacle, which has to “jump” from grid cell to grid cell of the Cartesian grid when moving. Despite both methods seem to be stable for this simulation scenario and chosen time step length, the explicit method shows larger oscillations than the implicit method, particularly for the forces.

## 6.2 Outflow and Slipwall Boundaries

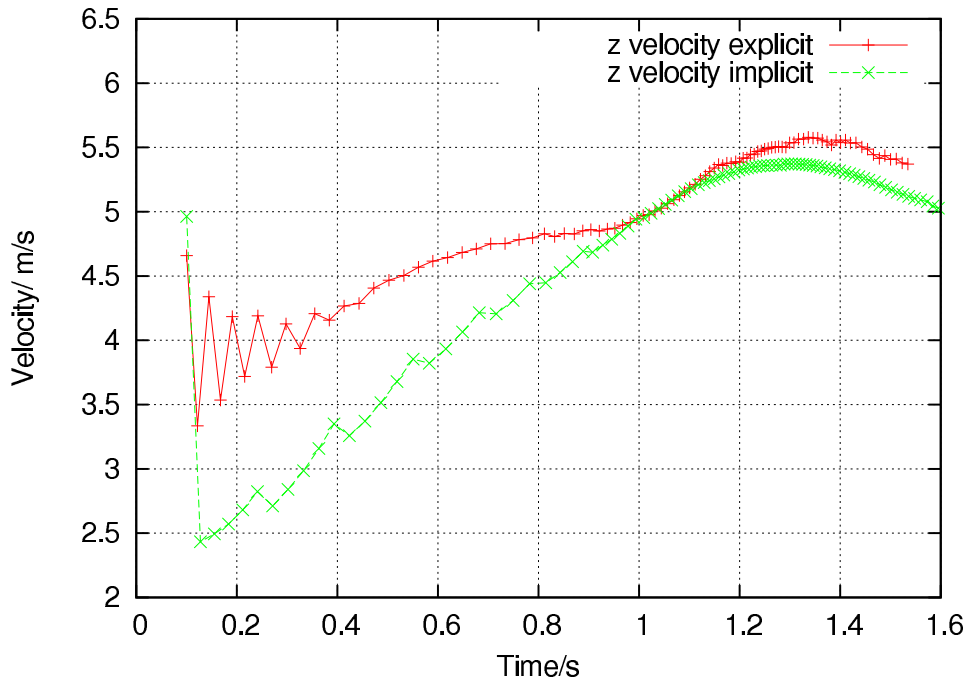
In Section 5.1, the extension of F3F with outflow and free-slip boundary types is described and in Section 5.3.2, the exchange of the convective and diffusive operators of F3F by operators from the simulation program PEANO. After these changes, tests were performed with empty channel flows using the new boundary types and operators.

To compare the results obtained by the new Neumann outflow boundary with that of the old Dirichlet boundary, results are shown for both boundary types in Figure 6.2. The flow direction is from front left to back right, the inflow profile is for both scenarios the same, while in (a) and (b) a outflow profile is prescribed (c) and (d) have the Neumann outflow condition, prescribing no direct profile at the outflow. While no differences can be visually observed for the velocity fields (a) and (c) of the flows, the pressure fields in

## 6 Numerical Results



(a) Forces on obstacle in stream direction.



(b) Obstacle velocity in stream direction.

Figure 6.1: Comparison of explicit and implicit time coupling scheme with internal FSI mode of F3F. The results are obtained from a channel flow with inserted spherical obstacle, being pushed stream-downwards by the flow. (a) shows the forces in stream direction and (b) the velocities in stream direction. While for both, the implicit and the explicit method, have jumps occurring every 3 to 5 steps (indicated by arrows in (a)), the implicit method shows significantly less oscillations.

(b) and (d) are different at the outflow of the domain. There, in (b), the pressure profile is non-uniform. This is due to the prescription of a 2d-parabolic outflow profile of the velocity, which is not the natural profile of a flow in a rectangular shaped channel<sup>1</sup>.

The results for the free-slip boundary conditions are shown in Figure 6.3. The flow direction is there, as indicated by the vectors in (a), from front right to back left. The scenario uses a Dirichlet inflow and a Neumann outflow boundary condition. The free-slip boundary conditions are applied to the front left and back right boundary, while the top and bottom boundaries have no-slip conditions. The velocity profile at the inflow is in one direction parabolic and in the other direction constant, as shown in Figure 5.1 on page 57. This leads to a constant velocity profile in one direction, and a parabolic velocity profile as observed in 2d flows in the other direction. The pressure field in (b) is ideally layered, since no disturbances of the applied inflow profile do occur in the flow.

## 6.3 FSI Benchmark

It was one major goal of this thesis to prepare the fluid solver F3F for computations of the FSI benchmarks. We will look at the description of the FSI benchmark scenario in Section 6.3.1 and see first results pointing into the right direction in Section 6.3.2.

### 6.3.1 Benchmark Description

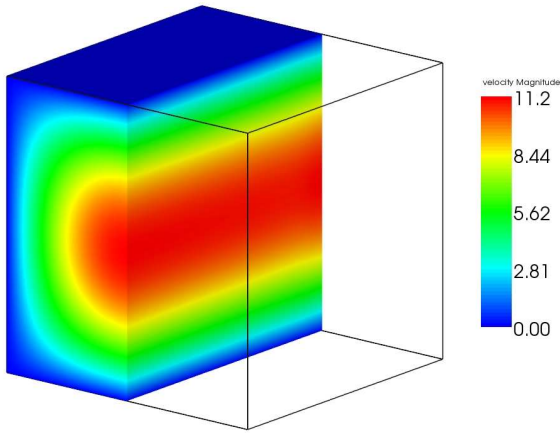
In [15], a so-called benchmarking scenario is proposed in order to evaluate the capability of an FSI simulation tool to produce quantitatively correct results. Some of these scenarios were used to test the maturity of the FSI simulation tools introduced within this work, namely the fluid simulation tool F3F coupled with the structure simulation tool AdhoC using the coupling environment FSIce. The most important characteristics of the FSI benchmark are described here, while the general setup of the scenario used in F3F to create a quasi-2d scenario have been already described in Section 5.1.2.

**Spatial Characteristics** The benchmark scenario is illustrated in Figure 6.4. It is a two-dimensional scenario with rectangular shape. Into the fluid domain, a cylindrical body with attached cantilever pointing in stream-wise direction is inserted. While the cylinder is fixed, the cantilever is free to move, with the restriction that the surface connected to the cylinder must be fixed. The obstacle is placed slightly non-centric in the channel to initiate controlled oscillations in the flow. Two locations  $A$  and  $B$  are used to compare

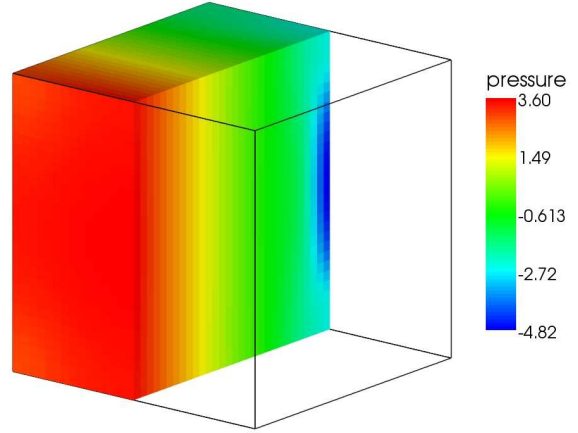
---

<sup>1</sup>The 2d-parabolic outflow profile is obtained, by applying a parabolic profile in one dimension normal to the flow direction and multiplying it with a parabolic profile in the second dimension normal to the flow direction. This velocity profile naturally occurs in cylindrical pipes.

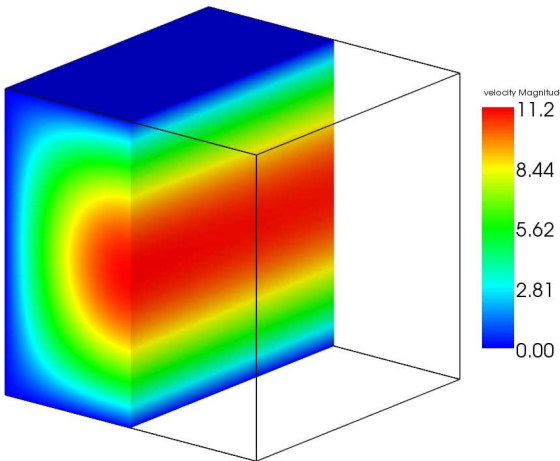
## 6 Numerical Results



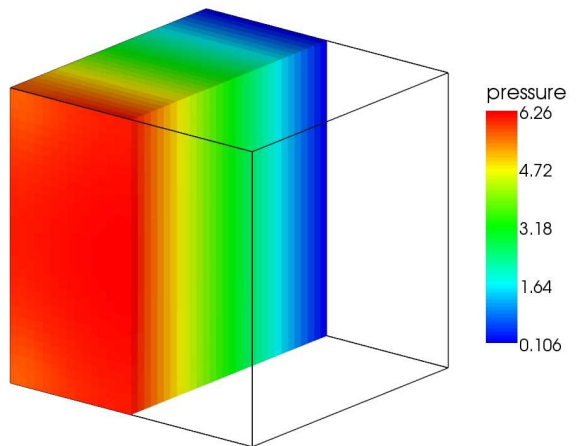
(a) Velocity field with Dirichlet outflow.



(b) Pressure field with Dirichlet outflow.



(c) Velocity field with Neumann outflow.



(d) Pressure field with Neumann outflow.

Figure 6.2: Empty channel flow with Dirichlet outflow condition in (a) and (b) and Neumann outflow condition in (c) and (d). The flow direction is from front left to back right. While the velocity fields (a) and (c) show no visually observable differences, the pressure fields (b) and (d) differ at the outflow of the domain. This is due to the enforced 2d-parabolic outflow profile in (b), which is not the natural profile of flows in rectangularly shaped channels. There, the pressure pushes the velocity profile into the enforced outflow profile. (This phenomenon is also observable in the pressure profile at the inflow in (b) and (d), but not so heavily, since the flow has the length of the channel to develop into its natural form.)

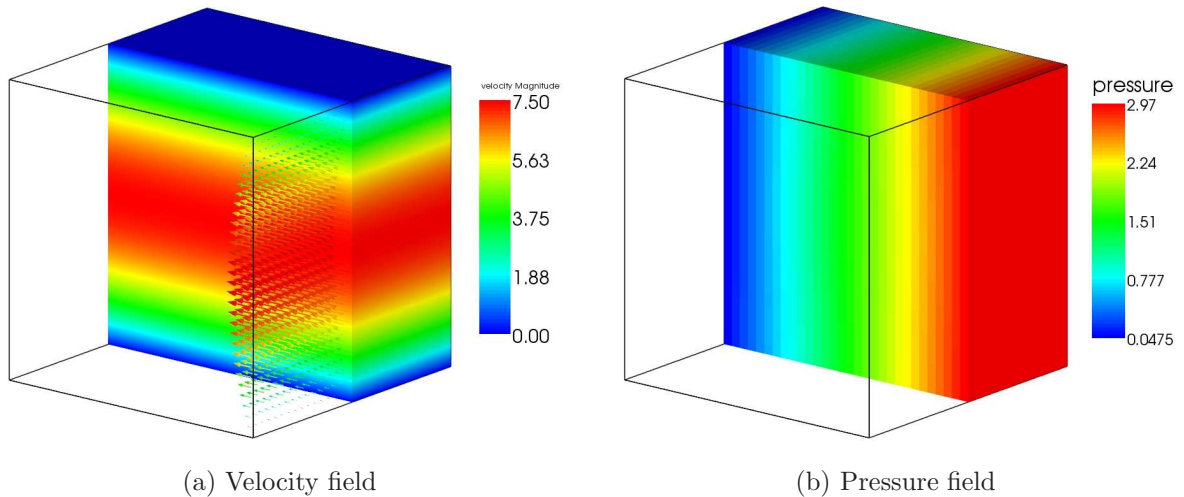


Figure 6.3: Empty channel flow with free-slip boundary conditions applied to the front left and back right boundary of the domain. Additionally, a Neumann outflow condition is applied at the outflow of the domain and no-slip boundary conditions are applied at the top and bottom boundary of the domain. Arrows in (a) indicate the flow direction, which is from front left to back right. The velocity profile applied at the inflow is parabolic in vertical direction and constant in horizontal direction. This inflow profile in combination with the boundary conditions leads to a flow which is constant in horizontal direction and parabolic in vertical direction. The inflow profile does not get disturbed by the boundary conditions, which manifests itself in a ideally layered pressure profile in (b).

characteristic quantities, where the point  $A = A(t)$  is bound to the movements of the cantilever and can hence give information about the oscillations of it.

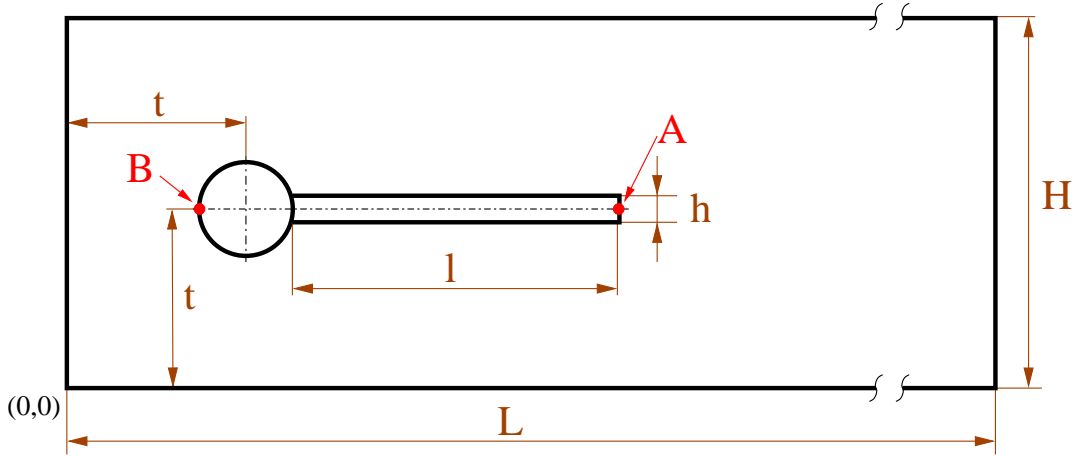


Figure 6.4: Sketch of the FSI benchmark scenario.

Quantitatively, the scenario is described by the following parameters

- Domain length  $L = 2.5$  and height  $H = 0.41$ .
- Cantilever length  $l = 0.35$  and height  $h = 0.02$ .
- Cylinder diameter  $d = 0.1$  (not marked in the figure).
- Cylinder center at  $(t, t)$  with  $t = 0.2$ .
- Coordinates point  $A(t = 0) = (0.6, 0.2)$ , point  $B = (0.15, 0.2)$ .

**Boundary Conditions** As boundary conditions, a dirichlet inflow condition with parabolic profile is applied as described in the benchmark. The outflow is a typical Neumann condition that implies no forces to the fluid. The sidewalls are no-slip Dirichlet walls, enforcing a zero velocity of the fluid.

**Quantities used for Comparison** In order to compare quantities of the fluid-structure interaction problem, one of course needs to define which quantities are used for comparison. In a first step, the time frame of measurement is determined. In the proposed benchmark, self induced oscillations are expected to develop and, thus, the time frame is set to be one full period of oscillation in the fully developed flow. The quantities of interest are given as:

1. Oscillations of the cantilever, measured by the displacement of the point  $A(t)$ .
2. Lift and drag forces acting on the whole obstacle body (cf. Remark 6.1). These forces are measured by the surface integral over the obstacle of the stress caused

by the fluid and given by

$$(F_{drag}, F_{lift}) = \oint \sigma ndS.$$

3. Pressure difference in-between the point  $B$  and  $A(t)$ , reading

$$\Delta p^{AB} = p^B - p^{A(t)}.$$

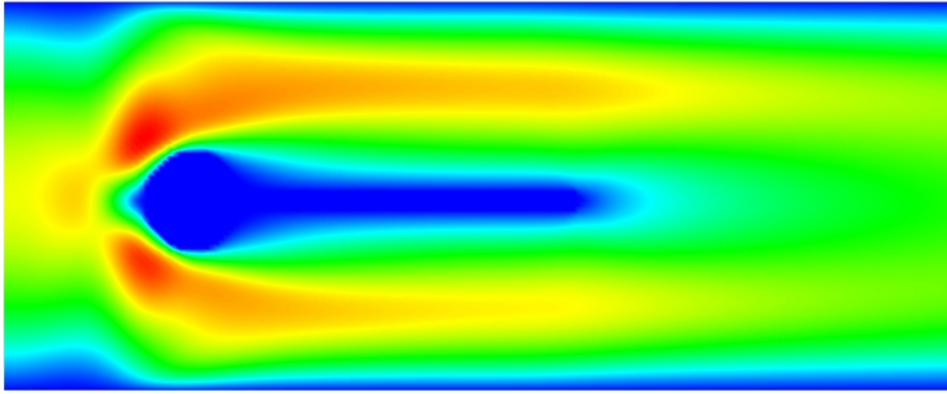
**Remark 6.1.** Lift and drag forces

In fluid dynamics, one speaks of lift and drag forces. These terms originate in the description of the forces acting on an airfoil, e.g. the wing of an airplane. Lift refers to the component of the force acting in flow-perpendicular direction, while drag identifies the component of the force acting in flow direction, i.e. in stream-wise direction.

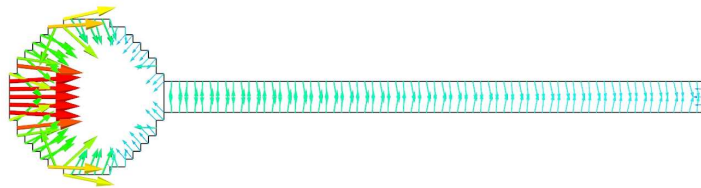
### 6.3.2 First Results

By the end of this thesis, the benchmark scenario was set up and several trial simulations were run. The results for the scenarios with steady flow are presented here. Figure 6.5 shows in (a) the qualitative velocity field of the front part of the scenario. Red colors indicate high velocities and blue colors low velocities. The forces acting on the obstacle are illustrated in (b). By summing up the total force over the surface of the obstacle, and scaling the resulting force by the factor  $s$  (equation (5.6)) as described in Section 5.1.2, one obtains forces comparable to the lift and drag reference forces  $F_{drag}$  and  $F_{lift}$  proposed in the FSI benchmark. In (c), the scaled drag force is shown. It converges to approximately  $15N$ , while the given reference value for  $F_{drag}$  of the FSI benchmark scenario is  $14.29N$ . Further simulations are currently under preparation, with results expected to come soon.

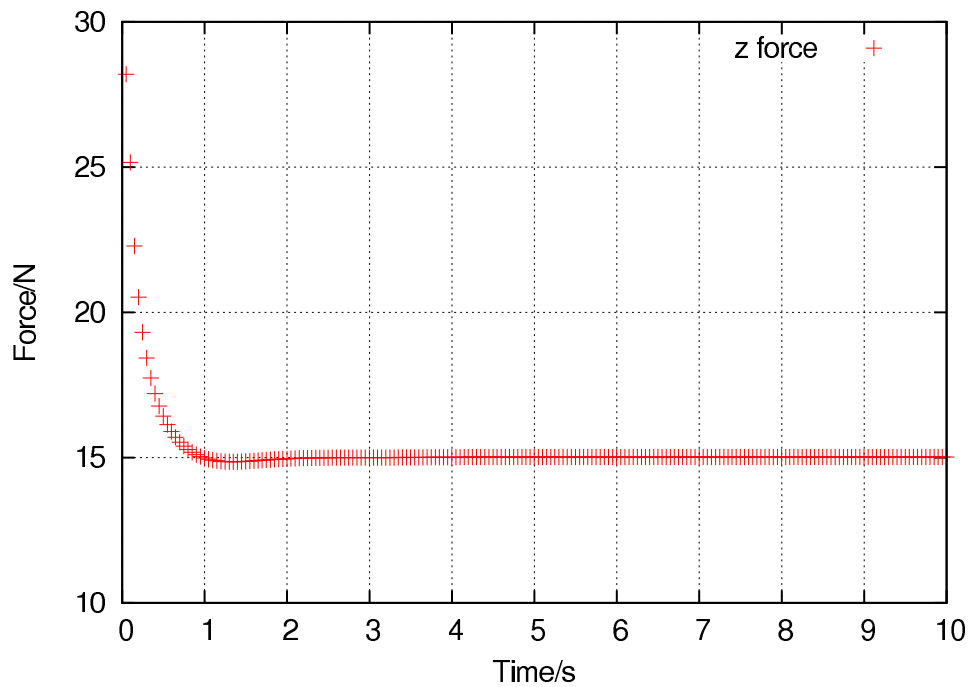
## 6 Numerical Results



(a) Qualitative velocity field of front part of FSI benchmark.



(b) Forces acting on the obstacle inserted in the flow.



(c) Properly scaled drag force  $F_{drag}$  to be compared with the benchmark results.

Figure 6.5: First results of the FSI benchmark scenario. In (c), the drag force  $F_{drag}$ , acting on the total submerged body of the obstacle in the flow is shown. The drag force converges to a value of  $15N$ , which is already close to the reference value of  $14.29N$ .

# 7 Conclusions

In this chapter, we will recall what has been achieved during this thesis and also critically evaluate these results (cf. Section 7.1). As not all ideas for a further improvement and enhancement of the codes could be implemented within a master's thesis, there is a large list of tasks that will be implemented in the near future. Thus, Section 7.2 lists several of these ideas, mainly concerning FSI\*ce.

## 7.1 What has been Achieved

In Chapter 2, the FSI coupling environment FSI\*ce has been introduced. The task of FSI\*ce can be summarized in one sentence: it has to offer a maximum amount of coupling related functionality and tools, while requiring only minimal effort from a user to integrate the coupling interface functionalities into his solver codes. To be able to fulfill this task, a proper software design and source code management is necessary, which at the same time, ensures the possibility of further development and maintainability of the code. The work done on FSI\*ce has to be measured with the requirements from the user and the developer side.

The work on FSI\*ce started with the restructuring of its source code, which was mainly driven by software development requirements. The source code project in its new form enables a more simple and faster development process and, in addition, speeds up the configuration and installation process for a user of FSI\*ce. The introduction of a dedicated package for the coupling supervisor prepared its future development, to implement more and more coupling functionality and provide it to the user. A part of this functionality has already been realized within this thesis. Subcycling, implicit coupling and pre-computations were integrated into the coupling supervisor and are no longer in the responsibility of a user of FSI\*ce. With the view of a software developer, the implemented coupling unit was written such, that it is independent from the concept of FSI, but allows any kind of multiphysics scenario. Following the work on the coupling supervisor, the application programming interface of FSI\*ce had to be modified, too, since the information exchange between solver and supervisor had to be increased to allow an encapsulation of more coupling functionality into the coupling supervisor. In addition, the interface has been extended by helper functionalities to save unnecessary computations on the solver side such as a repeated reading of not updated surface values, or an unnecessary transmission of surface values to the coupling supervisor. Not

only extensions have been made, parts of the interface have also been simplified. The separated call to send and receive surface values is now hidden behind one interface function, for example. All in all, the value of FSI\*ce has been improved in this thesis and future developments can be done with fewer efforts now.

The second part of this thesis was to prepare the fluid simulation tool for the computation of the FSI benchmark scenarios. This part of the work was not so straight forward as the work on FSI\*ce, but included many smaller often dislinkt modifications on F3F. Some of the major development work on F3F is described in Chapter 5 and we will recall this part here.

Since F3F cannot simulate the FSI benchmark directly, because it is bound to simulate three-dimensional scenarios only, a proper setup of a scenario had to be developed, allowing to compute comparable results to that of the two-dimensional benchmark. One major step to reach this goal was to implement new boundary types, namely outflow and free-slip boundaries. In that course, problems occurred and could be only solved (for the time being) by exchanging the convective and diffusive operators of F3F with those from another simulation program. Another set of functionality that needed improvement was the handling of multiple obstacles in F3F. A mechanism was derived, that ensures a proper setting of celltypes and nodetypes in the discretized domain of F3F. After the new application programming interface of FSI\*ce had been developed, it had to be integrated into F3F<sup>1</sup> to make use of it.

The preparations of F3F for the FSI benchmark are not yet completed, which explains why no final results of the benchmark scenarios are presented in this work. However, these results will be achieved in the near future.

## 7.2 Outlook on further Development

This outlook regards the development of FSI\*ce only and is given in the form of a list. The ordering of the items is not meant to impose any priority or importance to the listed items.

- Extension of available communication means (so far sockets and MPI) by grid environment conform technologies, in order to support distributed computing.
- Re-activation of the GUI viewer for the coupling supervisor, to enable “online” supervision of coupled simulations.
- Implementation of an XML-based configuration of the coupling supervisor, enabling it to centralize characterizing information for a coupled simulation, such as

---

<sup>1</sup>Additionally, the new interface functions had to be integrated into the structure solver AdhoC, which was the only “inside” contact with this program during the thesis. Fortunately, the new interface proved to be easily integrable.

## 7 Conclusions

synchronized output of information or creation of checkpoints of the solvers.

- Extension of logging capabilities of the coupling supervisor, acting as central logging unit for all participants of a coupled simulation.
- Upgrading the coupling mesh to support adaptive meshes.
- Providing support for multigrid solution procedures with mesh exchange on different grid levels.
- Implementation of more coupling schemes for the coupling supervisor.
- Implementation of more sophisticated measures for the convergence of the implicit coupling scheme.
- Implementation of convergence accelerators such as the Aitken acceleration technique for the implicit coupling scheme.

# A Explicit and Implicit methods for the numerical solution of ODEs

Our investigations in the world of ordinary differential equations (ODEs) shall be limited here to the simple case of the Cauchy or initial value problem for scalar functions. Our goal is to find a numerical approximation to the exact solution  $y \in C^1(I)$ , with  $I \subset \mathbb{R}$  as interval of interest, of the problem

$$\begin{aligned}y'(x) &= f(x, y(x)), & x \in I \\y(x_0) &= y_0, & x_0 \in I,\end{aligned}\tag{A.1}$$

where  $y(x_0)$  is called the initial value and  $f : I \times (-\infty, +\infty) \rightarrow \mathbb{R}$  is a real-valued function.

In order to numerically approximate the exact solution  $y$ , we try to solve the Cauchy problem by creating a series of approximations  $y_n$  at the discrete points  $x_n = x_0 + nh$ , with  $h$  being the discretization stepsize and  $n \in \mathbb{N}$ . The methods applicable to find this kind of solution can be divided into two categories, explicit and implicit. An explicit method is characterized by its exclusive use of previous approximations of  $y$ , i.e. all  $y_k$  for  $k \leq n$ , in the evaluation of the right-hand side  $f(x, y(x))$  for the computation of the next approximation  $y_{n+1}$ . An implicit method takes into account also approximations not yet computed and hence, wants to find a solution depending implicitly on itself through  $f$ . This initial part is inspired by the work of Quarteroni [21].

A simple example for a one-step explicit method is the forward Euler method. It computes the  $y_n$  by the rule

$$y_{n+1} = y_n + hf(x_n, y_n).\tag{A.2}$$

The same method exists also as implicit version, reading

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}).\tag{A.3}$$

In some special cases, the implicit method can be reordered for  $y_{n+1}$  and direct solutions can be found. Mostly, however, an iteration process has to be invoked for each computation step, approximating the value of  $y_{n+1}$  up to a certain accuracy.

Expanding  $y_{n+1}$  by Taylor expansion and inserting the expansion into either one of the Euler methods shows, that both methods have the same level of approximating accuracy,

which is  $O(h)$ . Hence, the question remains, why an implicit method is necessary at all, especially with concern to its higher computational costs. To justify the use of an implicit method, we look at a special example of the initial value problem, given by

$$f(x, y(x)) = -y, \quad y(0) = 5. \tag{A.4}$$

One reason why we choose this example is, that we can easily determine the analytical solution of it, which is

$$y(x) = 5e^{-x}. \tag{A.5}$$

In addition, we can compute the numerical approximations  $y_n$  directly by the implicit Euler method with

$$\begin{aligned} y_{n+1} &= y_n + hf_{n+1} = y_n - hy_{n+1} \\ \Leftrightarrow y_{n+1} &= \frac{y_n}{1+h}. \end{aligned}$$

We can now use the analytical solution (A.5) of the Cauchy problem (A.4) to compare the performance of our explicit Euler method (A.2) with its implicit counterpart (A.3). Figure A.1 shows the results of this comparison, for increasing stepsize  $h$ . In subfigure (a) with  $h = 0.5$ , both methods are equally close to the exact solution (A.5), denoted by the dashed line. The explicit Euler method stays below, while the implicit version stays above the exact solution. In subfigure (b) with  $h = 1.5$ , the explicit method starts to show some oscillations, but of declining nature. Subfigure (c) shows the border case for the explicit method at  $h = 2.0$ , stable oscillations develop, and the method is not converging to the exact solution. Finally, in subfigure (d) ( $h = 3.0$ ), the explicit method is diverging in increasing oscillations, while the implicit method still converges without any oscillations.

The observed behavior can be explained by some simple recursive analysis. We first look at the explicit Euler method (A.2). With inserted Cauchy problem (A.4), the approximation rule reads

$$y_{n+1} = y_n - hy_n = (1-h)y_n. \tag{A.6}$$

We can now expand  $y_n$  recursively by applying the same rule

$$\begin{aligned} y_{n+1} &= (1-h)(y_{n-1} - hy_{n-1}) \\ &= (1-h)(1-h)y_{n-1} \\ &\quad \vdots \\ y_{n+1} &= (1-h)^n y_0. \end{aligned} \tag{A.7}$$

Equation (A.7) gives an explanation for the behavior of the explicit Euler method when increasing  $n$ . Clearly, for stepsize  $h \leq 1$ , the approximations of  $y_n$  converge to the exact solution without oscillations. For  $1 < h < 2$ , convergence is still achieved, now with

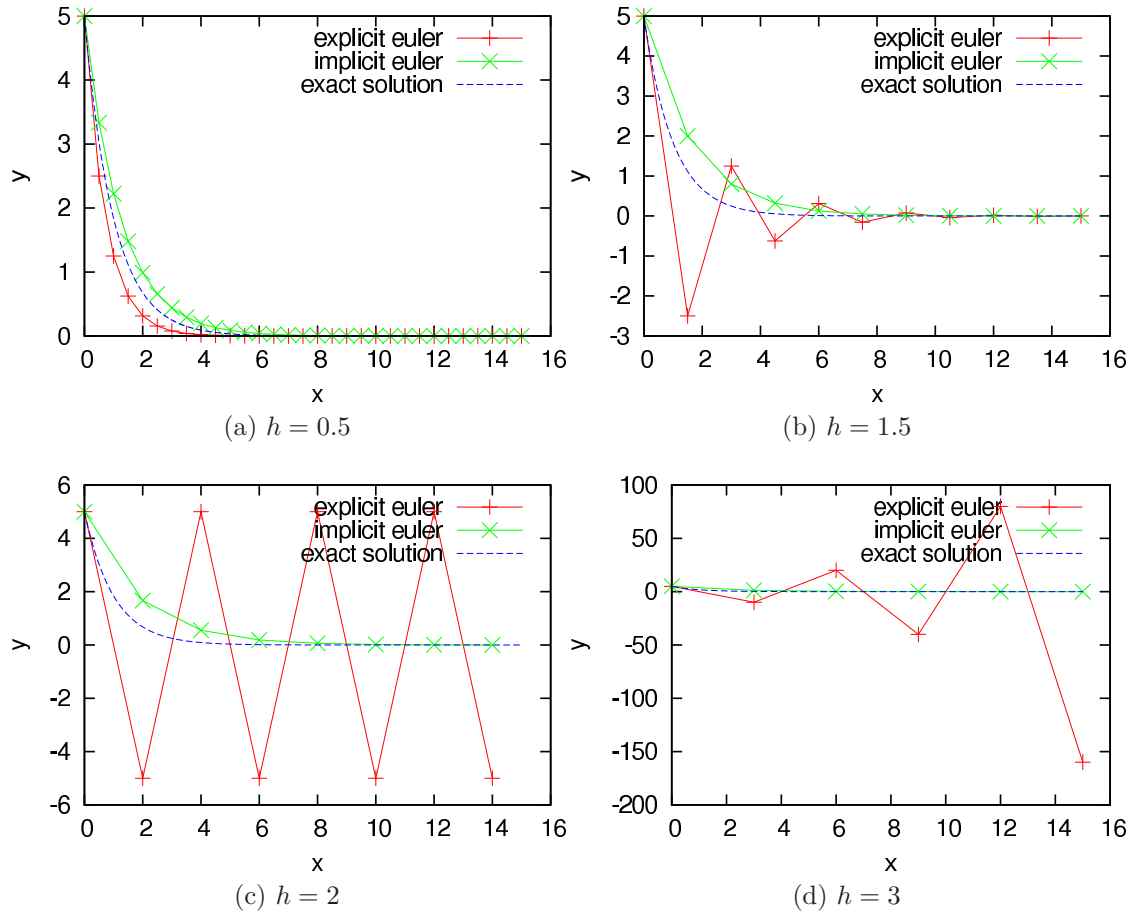


Figure A.1: Numerical approximation of the Cauchy problem (A.4) with different discretization step widths  $h$ . The explicit Euler method diverges for  $h > 2$ , while the implicit Euler method remains stable.

declining oscillations. The limit-case  $h = 2$  yields constant oscillations of amplitude  $y_0$ , and for  $h > 2$ , no convergence is achieved at all. Applying the same analysis to the implicit Euler method (A.3) yields

$$y_{n+1} = \frac{y_0}{(1+h)^n}, \quad (\text{A.8})$$

which is converging to the exact solution for any  $h$ , since the denominator is bigger than 1 (of course,  $h > 0$  must be true, which is to be expected in a numerical integration method).

The phenomena observed are due to a special property of our example problem (A.4), which is called “stiffness”. To solve a “stiff” problem by an explicit time stepping method, requires to limit the discretization stepsize used to a certain maximum limit, which is problem dependent. In our example problem (A.4), this limit was  $h = 2$ . It does not suffice, to obey this limit only for some, say very critical, integration steps.

Oscillations will immediately arise in any stage of the integration process, as soon as the maximal limit for  $h$  is passed. The prescribed time step limit might be so small, that the use of a per step more expensive implicit method is computationally cheaper, since it allows arbitrary big step widths.

To carry the lesson over to the world of coupled systems again: Here, for certain problems, similar phenomena as described can occur due to the separation of the subfields into partitions. For these kinds of problems, an implicit coupling scheme might be the best choice.

## B Introduction to Tensor Notation

For introducing the basic equations of fluid and structure mechanics, tensor notation is used in this work. Tensors are categorized according to their order, with the following orders appearing in this work<sup>1</sup>:

**0 order** → Scalar  $a$   
**1st order** → Vector  $a_i := \{a_i \mid \text{with } i = 1, \dots, d\}$   
**2nd order** → Matrix  $a_{ij} := \{a_{ij} \mid \text{with } i, j = 1, \dots, d\}$

The dimension of the tensors is given by  $d$ , with  $d = 3$  representing our well-known three dimensional world. Operations with tensors of order one, two, or three can be performed identically to scalar, vector or matrix operations respectively, as defined in linear algebra.

An important rule in tensor notation is the *Einstein summation convention*. It defines a contracted form for sums of tensors and is probably best explained with the help of a simple example, namely the divergence of vector  $a$

$$\frac{\partial a_i}{\partial x_i} := \frac{\partial a_1}{\partial x_1} + \frac{\partial a_2}{\partial x_2} + \frac{\partial a_3}{\partial x_3}, \quad \text{for } d = 3.$$

By this example we can deduce the rules of the Einstein summation convention. To indicate a sum in short form, a tensor index must appear twice within one group of tensors. In the example this is index  $i$ . A valid group must consist of tensors connected by  $\cdot$  or  $\div$ . Then, the expanded form of the tensor group is a sum of itself with indices being replaced by the actual numbers of the dimensions. Note, that if the group of tensors from the example would contain more tensors with differently named indices, these tensors would show up in the sum without replaced indices, as shown in the next example, the Laplacian of vector  $u$

$$\frac{\partial^2 a_i}{\partial x_j \partial x_j} := \frac{\partial^2 a_i}{\partial x_1^2} + \frac{\partial^2 a_i}{\partial x_2^2} + \frac{\partial^2 a_i}{\partial x_3^2}.$$

In this example, the tensor with index  $i$  is kept also in the expanded form, since the index  $i$  appears only once within its group. The index  $i$ , remaining in the expanded form, indicates, that this equation actually describes three equations, namely for  $i = 1, 2, 3$ .

---

<sup>1</sup>The elasticity tensor appearing in section 3.2.3 of this work is actually of order four. It is, however, not aimed to give a full understanding of its structure and meaning, which leaves it out of any discussion in this appendix.

# C F3F Code Details

This appendix is meant for a developer of F3F and intends to give him some hints on important changes in the code of it.

**Preprocessor switches** During the work on F3F, several switches to choose between old and new functionality were introduced in the code by using preprocessor `define` directives. The relevant ones are listed here:

- To switch between F3F and PEANO discretized Navier-Stokes operators, use `#define USE_PEANO_OPERATORS`, given in file `f3f_config.h`.
- To select proper force scaling and total force output for FSI benchmark scenarios, use `#define FSI_BENCHMARK`, given in file `f3f_config.h`.
- To switch between new and old coupling API and hence coupling supervisor, use `#define FSI_NEW_INTERFACE`, given in file `f3f_config.h`.
- To use FSI functionalities as implemented by Klaus Daubner, insert `#define FSI_KJD` to file `f3f_config.h` before statement `#ifndef FSI_KJD`.

**Inflow profile for DFG cylinder and FSI benchmark** The DFG cylinder and the FSI benchmark are both two-dimensional scenarios. A new inflow profile was developed during this work (cf. Section 5.1.2), that enables to simulate these benchmark and get comparable results. The type of the profile to be used in the XML-input file of F3F is called `fsibenchmark`. The input velocity given for the DFG cylinder benchmark and the FSI benchmark have different meanings. While the one in the cylinder benchmark refers to the maximal velocity of the inflow profile, the velocity used for the FSI benchmark refers to the mean inflow velocity. The current implementation is suitable to use the mean velocity as input parameter.

**Constants for node and cell types** Common constants for node and cell types were introduced during this thesis. The constants are added to the file `f3f_config.h` and should be used whenever node or cell types are queried.

**Helper macros for for loops** To simplify iterations over three- or two-dimensional array-like data structures helper macros were introduced in this thesis. The macros are gathered in the file `f3f_config.h` and are well documented. In addition, a helper macro to simplify iterations over list-like data structures by `iterators` was introduced in the same file.

# Bibliography

- [1] Mpccci 3.0: Multidisciplinary simulation through code coupling. <http://www.mpccci.de/>, 2007.
- [2] M. Bader, A. Frank, and C. Zenger. An octree-based approach for fast elliptic solvers. *High Performance Scientific and Engineering Computing*, 21:157–166, 2002.
- [3] L. M. Brekhovskikh and V. Goncharov. *Mechanics of Continua and Wave Dynamics*. Springer, 2005.
- [4] M. Brenk. *Algorithmic Aspects of Fluid-Structure Interactions on Cartesian Grids (German: Algorithmische Aspekte der Fluid-Struktur-Wechselwirkung auf kartesischen Gittern)*. PhD thesis, TU München, 2007.
- [5] M. Brenk, H.-J. Bungartz, I. L. Muntean, and T. Neckel. Simulating large particle movements in drift ratchets using cartesian grids. In P. Wesseling, E. Oñate, and J. Périaux, editors, *ECCOMAS COUPLED PROBLEMS 2007, Proc. of the Thematic Conf. on Computational Methods for Coupled Problems in Science and Engineering*, pages 397–399. International Center for Numerical Methods in Engineering (CIMNE), 2007.
- [6] M. Brenk, H.-J. Bungartz, and T. Neckel. Cartesian discretisations for fluid-structure interaction – consistent forces. In P. Wesseling, E. Oñate, and J. Périaux, editors, *ECCOMAS CFD 2006, European Conference on Computational Fluid Dynamics*. TU Delft, 2006.
- [7] A. Buck. Effiziente Randbehandlung unregelmäßiger Geometrien bei Simulation von Strömungen mit symmetrieerhaltender Diskretisierung. Diploma thesis, Fakultät Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart, 2005.
- [8] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22:745–762, 1968.
- [9] K. Daubner. Data exchange and geometry treatment for the simulation of fluid-structure interactions with partitioned approaches (german: Datenaustausch und geometriehandlung bei der simulation von fluid-struktur-wechselwirkungen mit partitionierten ansätzen). Diploma thesis, Institut für Informatik, TU München, 2005.
- [10] A. Düster. *High order finite elements for three-dimensional, thin-walled nonlinear continua*. PhD thesis, TU München, Fakultät für Bauingenieur- und Vermessungswesen, Chair for Computation in Engineering, 2001.

## Bibliography

- [11] M. Emans. *Numerische Simulation des unterkühlten Blasensiedens in turbulenter Strömung: Ein Euler-Lagrange-Verfahren auf orthogonalen Gittern*. PhD thesis, Institut für Informatik, TU München, 2003.
- [12] Carlos A. Felippa, K. C. Park, and Charbel Farhat. *Partitioned analysis of coupled mechanical systems*. 1999.
- [13] P. M. Gresho and R. L. Sani. *Incompressible flow and the finite element method*. Wiley, 1998.
- [14] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *Physics of fluids*, 8(12):2182–2189, 1965.
- [15] J. Hron and S. Turek. Proposal for numerical benchmarking of fluid-structure interaction between elastic object and laminar incompressible flow. In H.-J. Bungartz and M. Schäfer, editors, *Fluid-Structure Interaction*, number 53 in LNCSE, pages 371–385. Springer, 2006.
- [16] Free Software Foundation, Inc. GNU Autoconf - Creating Automatic Configuration Scripts. <http://www.gnu.org/software/autoconf/manual/autoconf.pdf>, 2008. Manual.
- [17] MPI: A message-passing interface standard, version 1.1, 1995. Manual.
- [18] MPI-2: Extensions to the message-passing interface, 1998. Manual.
- [19] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005.
- [20] I. Papaioannou. Fluid-structure interaction with high order finite element methods. Master’s thesis, TU München, Fakultät für Bauingenieur- und Vermessungswesen, Chair for Computation in Engineering, 2007.
- [21] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics, Second Edition*. Springer, 2007.
- [22] P. Wriggers. *Nichtlineare Finite-Elemente-Methode*. Springer, 2001.