An Explicit Method for Compressible Flow Simulation with Regards to Fluid-Motion Coupling

Jan Krautter

Born 11th March 1992 in Schorndorf

19th January 2016

Master's Thesis Mathematics

Prof. Dr. Michael Griebel

Prof. Dr. Marc Alexander Schweitzer

Institute for Numerical Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

Chapter 1

Introduction

The physical understanding of processes involving flowing gases and/or liquids, so-called fluids is a field of vast theoretical interest, as well as many practical applications. The corresponding differential equations however, still pose a huge challenge, as of today. Even in the most favourable conditions, they resist any attempt at proving global solutions and for complex problems, explicit solutions in closed form are out of the question, anyway. On the other hand, physical experiments, often conducted in huge wind or water channels, are very expensive and lack the kind of visualization and evaluation possibilities of mathematical solutions. Designing robust and accurate numerical methods to approximate solutions, and cover the middle ground between these two extremes, preferably establishing a positive feedback loop with either of them, thus, becomes all the more important. On the numerical side, flows around static geometries are by now comparatively well understood in both the incompressible, as well as the compressible case, at least in the laminar flow regime. For higher Reynolds numbers, turbulent phenomena still cause a lot of headache. The advent of ever faster computer hardware and techniques such as the large-eddy simulation, however, push the boundaries very fast in this respect, so it seems like it might only be a matter of time, until even very complex simulations of today's standard might be handled rather easily. There is a whole different class of fluid problems, though, that still has many theoretical, as well as practical challenges ahead, despite many interesting applications and the ever increasing interest in it. In the most general sense, we are talking about flows, involving geometries, that evolve over time, either by having some parts of the geometry that are moving and/or rotating, like with a turbine or an engine, or even more complex, by having flexible or otherwise deforming objects, whose shape is dependant on the flow surrounding them. The most prominent example, that showcases the importance of these sorts of problems, until today, remains the collapse of the Tacoma Narrows Bridge in 1940, due to unforeseen oscillations caused by such *fluid-structure interaction* (FSI). Another take on the difficulty of such

problems, is the slightly ironic fact, that a paper air plane is, at least in some regards, actually much more difficult to understand and simulate, than a real aircraft. One huge area of applications, that we will come back to, in the course of this thesis, is the biological world, most prominently represented by the flight of birds and/or insects. Quite obviously, a simulation solely based on static obstacle geometry can only describe such a complex process, in a very limited way.

For the course of this thesis, keeping these complex problems as a motivation in the back of our mind, we want to first look at more basic examples of this class, restricting ourselves to the first case of purely moving geometries, rather than a full fluid-structure interaction. Breaking such processes further down, we distinguish between a so-called unidirectional fluid-motion coupling, where only influences in one of the directions are considered, and a bidirectional coupling, were both are considered at the same time. In the first case, we are interested in the influences, an obstacle, with a prescribed trajectory, has on the surrounding flow, neglecting the forces exerted by the fluid on said obstacle. One can immediately think of many applications, like stirring a pot of water, where such a premise would make sense. The other case, were objects are passively transported and/or transformed by a flow also has some legitimate applications, but will not be discussed, here. Instead, we want to go ahead and tackle at least some simple cases of a bidirectional coupling, which captures either of the two effects. Many CFD (computational fluid dynamics) codes, that build on an incompressible fluid description, suffer from numerical stiffness leading to a badly conditioned system matrix, as well as a very sensitive dependence on initial conditions, when they are extended by such features. Instantaneous movement, as an example, sometimes leads to severe stability issues, and much care has to be taken to design experiments accordingly. The basic idea of this thesis, is hence, to consider (weakly-) compressible flows as the underlying basis for such an endeavour, and see if the mentioned problems can be addressed in this way, or if their handling can at least be improved, in this setting. We hope that the intrinsic damping effects, of this additional compressibility, might help our cause. Another way of looking at it, is that the finite propagation speed of pressure disturbances, that comes with a compressible fluid description, should improve the numerical stiffness encountered otherwise.

The pursuit of very dynamic cases of coupled problems, where a lot of movement goes on in the scene, is also an area, where a regular Cartesian grid can still play it's strength, compared to the very complicated mesh-based methods, that most proprietary CFD tools use. While such a mesh can be deformed to some degree, if the changes are too severe, eventually, an expensive re-meshing of the geometry is often required, and the earlier state

has to be mapped onto this new mesh. In our case, however, the obstacles just move in front of a fixed background grid. This also facilitates the potential for independent movement of multiple objects. Another benefit, which is closely tied to this, is the much easier handling of parallelization, and a better scaling behaviour, when utilizing many CPUs at the same time. The parallelization of the implemented algorithms shall be realized in the MPI (Message Passing Interface) framework, to enable the use of big compute clusters. Another goal of this work, is to allow for the trouble-free use of very complex geometries. A streamlined procedure, were one of the prevalent file formats for 3D geometry can be directly imported by the program is planned. This way, an external CAD software can be used to generate arbitrarily sophisticated models.

Accomplished contributions

The independent contributions to this work can be categorized in three areas and briefly summarized as follows:

Theory Selection of a suitable method to solve the compressible Navier-Stokes equations and subsequent combination with the level-set approach to represent geometry. Adaptation of the level set transport to the situation at hand.

Implementation Extension of the existing NaSt3DGP software by an explicit, compressible fluid solver, adaptation of corresponding output routines, import of stl-files and implementation of a point-in-polygon test to generate a level set function, routines to calculate fluid forces on a level set geometry and parallelization of the new modules with existing MPI routines.

Numerical simulation Simulation of various basic test cases, replication of a benchmark result, and presentation of experiments showcasing practical applicability and the capabilities in fluid-motion coupling.

Structure

The present thesis is divided into four main parts, followed by the conclusion. Before starting off, we want to give a short overview of the contents of each individual Chapter.

Chapter 2 In Chapter 2 we first give a description of the physical model, on which our numerical enquiries are based, before providing a detailed explanation of the utilized algorithms to approximate solutions of the governing equations.

Chapter 3 Chapter 3 is dedicated to the implementation of complex geometries in our code, which, in this case, are represented by a so-called level set function, and can be imported via the stl file format.

Chapter 4 Having static geometry in place, we look at the possibility of moving obstacles through our scene and different ways of coupling their motion with the fluid dynamics.

Chapter 5 In Chapter 5 we present some benchmark experiments in order to validate our approach, and demonstrate the potential of the presented algorithms, by showing some examples of both unidirectional, as well as bidirectional fluid-motion coupling.

Chapter 6 Finally, we conclude with a summary of the presented results and a subsequent outlook at further opportunities.

Chapter 2

An explicit compressible flow solver

The goal of this work is to implement an efficient, numerical fluid solver, which is well-adapted to solve problems with some kind of evolving geometry, in the simplest case, by moving a fixed, solid object at a given velocity throughout the domain. Due to the beneficial nature of damping effects, which are introduced by a certain compressibility of our medium, we opt for a weakly-compressible description of fluid flows, certainly in the subsonic flow regime. This is expected to help with the numerical stiffness encountered in some otherwise comparable, but incompressible methods. In order to put this into practice, we start out with a continuous description of the involved processes, which is given by partial differential equations. In a second step we try to discretise these equations on a given grid to approximate solutions.

2.1 The compressible Navier-Stokes equations

In this section, we want to describe our mathematical model for these compressible fluid flows. The equations we want to discuss are therefore the compressible Navier-Stokes equations. They were first stated in the mid-nineteenth century and can be derived solely from conservation principles, namely the conservation of momentum, mass and energy. In their most general form, they describe all sorts of viscous fluids, here we want to restrict ourselves to the following Newtonian form of the equations:

Let $\Omega \subset \mathbb{R}^3$ be a domain and $t \in (0, \infty)$ the time. Then our flow can be described by the velocity field $\mathbf{u} = (u, v, w) : \Omega \times (0, \infty) \to \mathbb{R}^3$ together with the scalar density $\rho : \Omega \times (0, \infty) \to \mathbb{R}$ and pressure $p : \Omega \times (0, \infty) \to \mathbb{R}$. In three dimensions, neglecting external body forces for the sake of clarity, the

compressible, isentropic Navier-Stokes equations can be stated as

$$\frac{\partial}{\partial t}(\rho \mathbf{u}) + \operatorname{div}(\rho \mathbf{u} \otimes \mathbf{u}) - \mu \Delta \mathbf{u} - (\lambda + \mu) \nabla \operatorname{div} \mathbf{u} + \nabla p = 0, \tag{2.1}$$

$$\frac{\partial}{\partial t}\rho + \operatorname{div}(\rho \mathbf{u}) = 0, \tag{2.2}$$

where the fluid properties μ and λ are called the dynamic and bulk viscosity, respectively, and are assumed to be constant in this frame work. We refer to (2.1) as the momentum equation and call (2.2) the continuity equation. To close the problem, an additional equation is required, which describes the connection between the pressure p and the density ρ, a so-called equation of state. In general, this relation is surely dependent on the temperature of the fluid, most notably if we are dealing with a gas. If there's no external heat entering, however, this effect is dependant on the Mach number, describing the ratio between the velocity of our fluid and the speed of sound, at which waves propagate through our fluid. In the low subsonic Mach number regime ($M << 1$), we can assume that the internal heat generation by friction is negligible and, we can thus postulate isothermal conditions. As a result we only consider a rather simple equation of state of the form

$$p = c^2 \rho, \tag{2.3}$$

where the proportionality constant c represents the speed of sound in the given medium and will hence be denoted v_{sound} throughout the rest of this work. For gases, this follows easily from the ideal gas law, if we set the Temperature T to a constant.

In the coordinates (u, v, w), this translates to the following set of equations:

$$\begin{aligned} &\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho v u)}{\partial y} + \frac{\partial(\rho w u)}{\partial z} \\ &\qquad - \mu \Delta u - (\lambda + \mu) \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) + \frac{\partial p}{\partial x} = 0, \end{aligned} \tag{2.4}$$

$$\begin{aligned} &\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho u v)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} + \frac{\partial(\rho w v)}{\partial z} \\ &\qquad - \mu \Delta v - (\lambda + \mu) \frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) + \frac{\partial p}{\partial y} = 0, \end{aligned} \tag{2.5}$$

$$\begin{aligned} &\frac{\partial(\rho w)}{\partial t} + \frac{\partial(\rho u w)}{\partial x} + \frac{\partial(\rho v w)}{\partial y} + \frac{\partial(\rho w^2)}{\partial z} \\ &\qquad - \mu \Delta w - (\lambda + \mu) \frac{\partial}{\partial z}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) + \frac{\partial p}{\partial y} = 0, \end{aligned} \tag{2.6}$$

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} = 0. \tag{2.7}$$

Throughout the rest of this work, in addition to the constant dynamic viscosity, we restrict ourselves to a fixed value of $\lambda = -\frac{2}{3}\mu$, which is a common assumption (see [Bat00]) and has the nice side-effect of giving us a traceless tensor for the shear-stress. For a full derivation of the depicted equations, we refer to the book of Gurtin [Gur81], but any proper textbook on continuum mechanics serves the purpose.

2.2 A short review of existing methods

Having our physical model fixed, we now want to discretise the involved equations to eventually solve them on a compute cluster. Before stating our final approach, we first want to give a very coarse overview of some existing methods, to allow for a better understanding of the reasons for the inevitable choices, involved.

Today, numerical methods for solving the compressible Navier-Stokes equations divide into finite difference, finite volume and finite element methods. Among these, finite differencing methods were the first to show up. Starting in the 60's the first explicit time integration methods were developed, leading among others to the large class of Lax-Wendroff type methods [LW60]. In the beginning of the 70's their implicit counterparts soon followed, most notably by Briley and McDonald [BM77], as well as Beam and Warming [BW78], whose method MacCormack in his *A Perspective of a Quarter Century of CFD Research* [Mac93] termed the "work horse method for viscous compressible flow". While explicit methods simply work by an update rule, these implicit methods solve a system of linear equations. In general implicit methods have advantages regarding broad applicability, due to their inherent stability characteristics, which often can be made independent of time steps. Explicit methods on the other hand, almost always have a tight restriction on the possible time step, but within their working domain, such methods can make for a very efficient tool, since the lack of assembling and solving a huge system matrix gives them a decent head start in performance. Finally, during the 80's, with the surfacing of van Leer's second order MUSCL scheme [Lee79] and other developments, finite volume methods became very popular. Derivatives of this method, like the *Kurganov and Tadmor (KT) central scheme* [KT00] are still among the most popular techniques used in today's CFD codes. Although finite element methods have taken the field in many other areas, and the attention to FEM seems to grow in recent times (see i.e. [Wen08]), in computational fluid dynamics, methods based on finite differences and finite volumes are still the most widespread approach for general solvers, because they typically pose less requirements on memory and computing time and admit less complex implementations [Wes01].

A major role in differentiating techniques further, plays the character of the underlying grid, on which the discretisation takes place. Non-uniform grids have many advantages in representing complex geometries and are best suited to finite volume approaches, due to their inherent integral formulation. Differencing schemes, on the other hand face more problems, when generalized to such complex grids. On a rectangular grid, however, the two methods perform pretty comparable. Such a simple grid has the advantage of easier and more efficient parallelization techniques, as well as the much better handling of changing geometry, since no re-meshing is required. The grid just stays as a permanent background, on which you draw your objects. To overcome the huge number of cells needed to represent thin and/or complicated geometries, often so-called adaptive grids are used, which are only refined in a certain predefined region in space. Since this can only have the purpose of giving a rough overview, and is in no way able to cope with the multitude of different tools, that have been developed during the past decades, we refer to the book of Ferziger and Perić [FP08] for a more in-depth overview of the general principles and different methods of computational fluid dynamics, as well as the review papers [Mac85] and [Mac93] by MacCormack for a historical perspective on said developments.

The NaSt3DGP fluid solver The numerical implementation of the presented work is based on the existing NaSt3DGP fluid solver. NaSt3DGP was developed by the research group in the Division of Scientific Computing and Numerical Simulation at the University of Bonn [GDN98]. It is a parallel three-dimensional fluid solver using finite differences to discretise the incompressible Navier-Stokes equations on a staggered grid. The solver is formally based on Chorin's projection method [Cho68]. The representation of obstacles is realized via a so-called flag-field, which provides the possibility to enable different properties of cells by binary shifts. Parallelization is handled with the MPI library and allows the deployment of the code on huge compute clusters. For more detailed information about the implementation and underlying numerics, we refer to [GDN98] and [Cro02].

The chosen method With the existing infrastructure of the NaSt3DGP package, as well as the limited time frame of this thesis in mind, we decided to go for maximum flexibility on the one hand, combined with a relatively simple approach to begin with, and see how far we can get, later on. Therefore, we initially opted for a fully explicit finite difference discretisation on the back of a uniform grid, with the option of adding some kind of adaptivity, eventually. Especially the advantages in parallelization, when running on ever growing compute clusters, paired with the versatility in representing obstacles, due to the lack of needing complicated meshing algorithms, played a major role in this decision. We started off with the simple flag-field implementation and

then extended it with the level set method, to have a better approximation of surface normals. In total this gives us a very solid and robust way of handling complex geometry, especially when foreseeing the possibility of looking at fluid-structure interaction (FSI) in the future. Compared to the incompressible NaSt solver, we reverted back to a collocated grid, since the staggered setup loses it's key benefits, in combination with compressible schemes and introduces a lot of otherwise unnecessary interpolation into the code. For the parallelization we try to reuse as much as possible of the existing routines, which should give us a solid basis on this front.

2.3 The MacCormack scheme in fluid dynamics

We want to discretise equations $(2.4) - (2.7)$ now on a rectangular grid using finite differences. For this purpose we employ a predictor/corrector scheme suggested in [PH06], albeit generalized to three dimensions. Perrin and Hou apply the scheme to the simulation of particulate flows. In our case, we want to explore the applicability of this method to more complicated objects, and thus restrict ourselves to the simulation of a single moving part of the geometry, although in theory, the employed approach could easily be adapted to multiple objects, as well. The scheme was first published in 1969 by MacCormack in his seminal paper [Mac03]. Later, in 1982, further developed the scheme in [Mac82] into a mixed implicit/explicit method, trying to combine advantages of both techniques. For the purpose of this thesis, however, we want to stay with his original, fully explicit approach from 1969 and see what we can achieve with it, when combined with a modern approach to represent geometry and on the basis of present-day hardware. For equations of the form

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{E}(\mathbf{U})}{\partial x} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial y} + \frac{\partial \mathbf{G}(\mathbf{U})}{\partial z} = 0, \tag{2.8}$$

the MacCormack scheme consists of the predictor step:

$$\begin{aligned} \mathbf{U}^{\star}_{i,j} = \mathbf{U}^{n}_{i,j} &- \frac{\Delta t}{\Delta x}(\mathbf{E}^{n}_{i+1,j,k} - \mathbf{E}^{n}_{i,j,k}) \\ &- \frac{\Delta t}{\Delta y}(\mathbf{F}^{n}_{i,j+1,k} - \mathbf{F}^{n}_{i,j,k}) - \frac{\Delta t}{\Delta z}(\mathbf{G}^{n}_{i,j,k+1} - \mathbf{G}^{n}_{i,j,k}), \end{aligned} \tag{2.9}$$

and the corrector step:

$$\begin{aligned} \mathbf{U}^{n+1}_{i,j} = \frac{1}{2}(\mathbf{U}^{n}_{i,j} + U^{\star}_{i,j} &- \frac{\Delta t}{\Delta x}(\mathbf{E}^{n}_{i,j,k} - \mathbf{E}^{n}_{i-1,j,k}) \\ &- \frac{\Delta t}{\Delta y}(\mathbf{F}^{n}_{i,j,k} - \mathbf{F}^{n}_{i,j-1,k}) \\ &- \frac{\Delta t}{\Delta z}(\mathbf{G}^{n}_{i,j,k} - \mathbf{G}^{n}_{i,j,k-1})), \end{aligned} \tag{2.10}$$

where $\mathbf{U} = (\rho, \rho u, \rho v, \rho w)$ in our case. This method is fully explicit, no system of equations needs to be solved. Note that we used forward differences throughout the predicting step and backward differences in the corrector phase. In principal this scheme is accurate up to second-order in both time and space. Applying this to our equations and inserting the equation of state, we deduce the following update rule:

$$(\rho)^{\star}_{i,j,k} = (\rho)^{n}_{i,j,k} - c_1\partial_i^+(\rho u))^n_{i,j,k} - c_2\partial_j^+(\rho v))^n_{i,j,k} - c_3\partial_k^+(\rho w))^n_{i,j,k} \quad (2.11)$$

$$\begin{aligned}(\rho u)^{\star}_{i,j,k} = (\rho u)^{n}_{i,j,k} &- c_1\partial_i^+(\rho u^2 + c^2\rho)^n_{i,j,k} - c_2\partial_j^+(\rho v u)^n_{i,j,k} \\ &- c_3\partial_k^+(\rho w u)^n_{i,j,k} + (2\mu+\lambda)c_4\partial_i^+\partial_i^-(u)^n_{i,j,k} \\ &+ \mu c_5\partial_j^+\partial_j^-(u)^n_{i,j,k} + \mu c_6\partial_k^+\partial_k^-(u)^n_{i,j,k} \\ &+ (\mu+\lambda)c_7(\partial_i^o(\partial_j^o v^n_{i,j,k}) \\ &+ (\mu+\lambda)c_8(\partial_i^o(\partial_k^o w^n_{i,j,k}),\end{aligned} \quad (2.12)$$

$$\begin{aligned}(\rho v)^{\star}_{i,j,k} = (\rho v)^{n}_{i,j,k} &- c_1\partial_i^+(\rho u v)^n_{i,j,k} - c_2\partial_j^+(\rho v^2 + c^2\rho)^n_{i,j,k} \\ &- c_3\partial_k^+(\rho w v)^n_{i,j,k} + \mu c_4\partial_i^+\partial_i^-(v)^n_{i,j,k} \\ &+ (2\mu+\lambda)c_5\partial_j^+\partial_j^-(v)^n_{i,j,k} + \mu c_6\partial_k^+\partial_k^-(v)^n_{i,j,k} \\ &+ (\mu+\lambda)c_7(\partial_j^o(\partial_i^o u^n_{i,j,k}) \\ &+ (\mu+\lambda)c_9(\partial_j^o(\partial_k^o w^n_{i,j,k}),\end{aligned} \quad (2.13)$$

$$\begin{aligned}(\rho w)^{\star}_{i,j,k} = (\rho w)^{n}_{i,j,k} &- c_1\partial_i^+(\rho u w)^n_{i,j,k} - c_2\partial_j^+(\rho v w)^n_{i,j,k} \\ &- c_3\partial_k^+(\rho w^2 + c^2\rho)^n_{i,j,k} + \mu c_4\partial_i^+\partial_i^-(w)^n_{i,j,k} \\ &+ \mu c_5\partial_j^+\partial_j^-(w)^n_{i,j,k} + (2\mu+\lambda)c_6\partial_k^+\partial_k^-(w)^n_{i,j,k} \\ &+ (\mu+\lambda)c_8(\partial_k^o(\partial_i^o u^n_{i,j,k}) \\ &+ (\mu+\lambda)c_9(\partial_k^o(\partial_j^o v^n_{i,j,k})\end{aligned} \quad (2.14)$$

and for the corrector step we get

$$\begin{aligned}2(\rho)^{n+1}_{i,j,k} = (\rho)^{n}_{i,j,k} + (\rho)^{\star}_{i,j,k} &- c_1\partial_i^-(\rho u))^{\star}_{i,j,k} \\ &- c_2\partial_j^-(\rho v))^{\star}_{i,j,k} - c_3\partial_k^-(\rho w))^{\star}_{i,j,k}\end{aligned} \quad (2.15)$$

$$\begin{aligned}2(\rho u)^{n+1}_{i,j,k} = (\rho u)^{n}_{i,j,k} + (\rho u)^{\star}_{i,j,k} &- c_1\partial_i^-(\rho u^2 + c^2\rho)^{\star}_{i,j,k} - c_2\partial_j^-(\rho v u)^{\star}_{i,j,k} \\ &- c_3\partial_k^-(\rho w u)^{\star}_{i,j,k} + (2\mu+\lambda)c_4\partial_i^+\partial_i^-(u)^{\star}_{i,j,k} \\ &+ \mu c_5\partial_j^+\partial_j^-(u)^{\star}_{i,j,k} + \mu c_6\partial_k^+\partial_k^-(u)^{\star}_{i,j,k} \\ &+ (\mu+\lambda)c_7(\partial_i^o(\partial_j^o v^{\star}_{i,j,k}) \\ &+ (\mu+\lambda)c_8(\partial_i^o(\partial_k^o w^{\star}_{i,j,k}),\end{aligned} \quad (2.16)$$

$$\begin{aligned}2(\rho v)^{n+1}_{i,j,k} = (\rho v)^{n}_{i,j,k} &+ (\rho v)^{\star}_{i,j,k} - c_1\partial_i^-(\rho uv)^{\star}_{i,j,k} - c_2\partial_j^-(\rho v^2 + c^2\rho)^{\star}_{i,j,k} \\ &- c_3\partial_k^-(\rho wv)^{\star}_{i,j,k} + \mu c_4\partial_i^+\partial_i^-(v)^{\star}_{i,j,k} \\ &+ (2\mu+\lambda)c_5\partial_j^+\partial_j^-(v)^{\star}_{i,j,k} + \mu c_6\partial_k^+\partial_k^-(v)^{\star}_{i,j,k} \\ &+ (\mu+\lambda)c_7(\partial_j^o(\partial_i^o u^{\star}_{i,j,k}) \\ &+ (\mu+\lambda)c_9(\partial_j^o(\partial_k^o w^{\star}_{i,j,k}),\end{aligned} \tag{2.17}$$

$$\begin{aligned}2(\rho w)^{n+1}_{i,j,k} = (\rho w)^{n}_{i,j,k} &+ (\rho w)^{\star}_{i,j,k} - c_1\partial_i^-(\rho uw)^{\star}_{i,j,k} - c_2\partial_j^-(\rho vw)^{\star}_{i,j,k} \\ &- c_3\partial_k^-(\rho w^2 + c^2\rho)^{\star}_{i,j,k} + \mu c_4\partial_i^+\partial_i^-(w)^{\star}_{i,j,k} \\ &+ \mu c_5\partial_j^+\partial_j^-(w)^{\star}_{i,j,k} + (2\mu+\lambda)c_6\partial_k^+\partial_k^-(w)^{\star}_{i,j,k} \\ &+ (\mu+\lambda)c_8(\partial_k^o(\partial_i^o u^{\star}_{i,j,k}) \\ &+ (\mu+\lambda)c_9(\partial_k^o(\partial_j^o v^{\star}_{i,j,k}),\end{aligned} \tag{2.18}$$

where the coefficients c_1 up to c_9 are defined as follows

$$\begin{aligned}&c_1 = \frac{\Delta t}{\Delta x}, c_2 = \frac{\Delta t}{\Delta y}, c_3 = \frac{\Delta t}{\Delta z}, \\ &c_4 = \frac{\Delta t}{(\Delta x)^2}, c_5 = \frac{\Delta t}{(\Delta y)^2}, c_6 = \frac{\Delta t}{(\Delta z)^2}, \\ &c_7 = \frac{\Delta t}{4\Delta x\Delta y}, c_8 = \frac{\Delta t}{4\Delta x\Delta z} \text{ and } c_9 = \frac{\Delta t}{4\Delta y\Delta z}.\end{aligned} \tag{2.19}$$

As mentioned by Tannehill et al. in [THR76] the deployment of forward/-backward differences in the predictor/corrector steps, respectively, can be interchanged. This can be done independently in each of the three spatial directions, so in total we end up with eight different possibilities. Tannehill et. al suggest to cycle them among consecutive time steps to eliminate asymmetries in the algorithm. We adopted this practice to our case. The detailed arrangement of the schemes is shown in table 2.1 below, which is adopted from [THR76]. Another advantage of this cyclic approach is, that these variants all have different Eigenvalues in a given Fourier component, which are the basis of deducing the stability condition, which we will discuss in Section 2.5. By cycling among them, we expect a smaller amplification of any one of these Eigenvalues, and therefore an improved behaviour in terms of stability.

2.4 Boundary conditions

With the update routines for the inner grid points being established, we now want to take a closer look at some possible boundary conditions. To do this, we first want to recall the situation we have at the boundary: We chose to discretise our equations on a collocated grid, meaning that all relevant

Table 2.1: Applied differencing sequence for the MacCormack scheme; modelled on the corresponding table in [THR76]

step	pred. $\Delta x/\Delta y/\Delta z$	corr. $\Delta x/\Delta y/\Delta z$
1	F/F/F	B/B/B
2	B/B/F	F/F/B
3	F/F/B	B/B/F
4	B/F/B	F/B/F
5	F/B/F	B/F/B
6	B/F/F	F/B/B
7	F/B/B	B/F/F
8	B/B/B	F/F/F

F forward difference, B backward difference.

quantities (u, v, w, ρ) are located on the same points of a fixed grid. The physical boundary will be placed exactly at the location of our boundary points, so basically no interpolation between different points is needed.

Velocity boundary conditions

As a result of the choices mentioned above, implementing Dirichlet boundary conditions is easy, as long as the geometry coincides with the grid. We just set the components of $\mathbf{u}$ to the desired values. Most commonly we will have

$$u_{i,j,k} = v_{i,j,k} = w_{i,j,k} = 0, \tag{2.20}$$

representing a wall with no-slip conditions located at grid point (i, j, k).

Since in the case of a curved object, our real geometry will deviate from these boundary points, we can apply a Taylor expansion to approximate the velocity u_{sur} at the actual boundary point $(\tilde{x}, \tilde{y})$, which is closest to the given grid point, if we have access to the outer normal $\boldsymbol{n}$ of the geometry. Using the abbrevations $\Delta\tilde{x} = |\tilde{x} - x|$ and $\Delta\tilde{y} = |\tilde{y} - y|$ we have

$$u_{sur} \approx u_{i,j,k} + \Delta\tilde{x}(u_x)_{i,j,k} + \Delta\tilde{y}(u_y)_{i,j,k} + \Delta\tilde{z}(u_z)_{i,j,k}. \tag{2.21}$$

Reorganizing the terms and substituting the appropriate one-sided finite differences, i.e.

$$\begin{aligned}(u_x)_{ijk} &= \frac{1}{2\Delta x}(-u_{i+2,j,k} + 4u_{i+1,j,k} - 3u_{i,j,k}) + O(\Delta x^2),\\ (u_y)_{ijk} &= \frac{1}{2\Delta y}(-u_{i,j+2,k} + 4u_{i,j+1,k} - 3u_{i,j,k}) + O(\Delta y^2),\\ (u_z)_{ijk} &= \frac{1}{2\Delta z}(-u_{i,j,k+2} + 4u_{i,j,k+1} - 3u_{i,j,k}) + O(\Delta z^2),\end{aligned} \tag{2.22}$$

we get the update rule, exemplary to the first octant of the boundary:

$$\begin{aligned}u_{i,j,k} = \frac{1}{c}\Big[&-\Delta\hat{x}\Delta y\Delta z(-u_{i+2,j,k} + 4u_{i+1,j,k})\\ &-\Delta x\Delta\hat{y}\Delta z(-u_{i,j+2,k} + 4u_{i,j+1,k})\\ &-\Delta x\Delta y\Delta\hat{z}(-u_{i,j,k+2} + 4u_{i,j,k+1}) + 2\Delta x\Delta y\Delta z(u_{sur})\Big],\end{aligned} \tag{2.23}$$

with $c = 2\Delta x\Delta y\Delta z - 3\left[\Delta\hat{x}\Delta y\Delta z + \Delta x\Delta\hat{y}\Delta z + \Delta x\Delta y\Delta\hat{z}\right]$.

All other cases work analogously, by switching the appropriate signs and will only be shown once, here: Let (n_x, n_y, n_z) be the outer normal to our obstacle and $(\sigma_x, \sigma_y, \sigma_z) = (\sigma(n_x), \sigma(n_y), \sigma(n_z))$, accordingly, the signum of it. Denote with Δ^s a signed version of the cell dimensions, i.e.

$$\Delta^s x = \sigma_x\Delta x, \quad \Delta^s y = \sigma_y\Delta y, \quad \Delta^s z = \sigma_z\Delta z, \tag{2.24}$$

then our update rule in the general case can be written as

$$\begin{aligned}u_{i,j,k} = \frac{1}{c}\Big[&-\Delta\hat{x}\Delta^s y\Delta^s z(-u_{i+2\sigma_x,j,k} + 4u_{i+\sigma_x,j,k})\\ &-\Delta^s x\Delta\hat{y}\Delta^s z(-u_{i,j+2\sigma_y,k} + 4u_{i,j+\sigma_y,k})\\ &-\Delta^s x\Delta^s y\Delta\hat{z}(-u_{i,j,k+2\sigma_z} + 4u_{i,j,k+\sigma_z})\\ &+2\Delta^s x\Delta^s y\Delta^s z(u_{sur})\Big],\end{aligned} \tag{2.25}$$

with now $c = 2\Delta^s x\Delta^s y\Delta^s z - 3\left[\Delta\hat{x}\Delta^s y\Delta^s z + \Delta^s x\Delta\hat{y}\Delta^s z + \Delta^s x\Delta^s y\Delta\hat{z}\right]$.

For the sake of clarity, the general case will be omitted throughout this chapter and we will only look at the special case above. The components v and w work exactly the same, by replacing u with the according symbol and will be omitted, too, when this is evident. We expect to get a smoother velocity near the boundary, using this boundary condition in the case of curved and/or more complex geometries. For a direct comparison of the two methods in the 2D case see [PH06].

Prescribed inflow/outflow with a fixed velocity. Additionally, we also want to have the possibility to simulate boundaries, with a prescribed inflow value, as well as free inflow/outflow type boundary conditions. These can be implemented as follows: Inflow is just another variant of the above, where now the values are different from zero. An open inflow/outflow condition is represented by the equation

$$\frac{\partial(\rho u)}{\partial x} = \frac{\partial(\rho v)}{\partial y} = \frac{\partial(\rho w)}{\partial z} = 0. \tag{2.26}$$

Finally we implemented symmetry boundary conditions.

Density boundary conditions

Deducing suitable boundary conditions for the density is more difficult. If we propose a fixed velocity at the boundary, a corresponding density should in theory be given by the equation. [PH06] suggest two different possibilities: one given by the continuity equation and another one based on the momentum equation. Since the one coming from the momentum equation seems favourable, we will adopt it to our case, in the following.

Let $\boldsymbol{n}$ be the outer normal to the geometry surface, then using (2.4) - (2.6) we can express the normal derivative of the density by:

$$\begin{aligned}
\partial_{\boldsymbol{n}}\rho = &\frac{n_x}{c^2}\left[\mu(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2})+(\mu+\lambda)(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 v}{\partial x\partial y}+\frac{\partial^2 w}{\partial x\partial z})\right]\\
&+\frac{n_y}{c^2}\left[\mu(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2})+(\mu+\lambda)(\frac{\partial^2 u}{\partial y\partial x}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 w}{\partial y\partial z})\right]\\
&+\frac{n_z}{c^2}\left[\mu(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2})+(\mu+\lambda)(\frac{\partial^2 u}{\partial z\partial x}+\frac{\partial^2 v}{\partial z\partial y}+\frac{\partial^2 w}{\partial z^2})\right]\\
&-\frac{\rho n_x}{c^2}\left[\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}\right]\\
&-\frac{\rho n_y}{c^2}\left[\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}\right]\\
&-\frac{\rho n_z}{c^2}\left[\frac{\partial w}{\partial t}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}\right],
\end{aligned} \tag{2.27}$$

or using A and B accordingly:

$$\partial_{\boldsymbol{n}}\rho = \frac{1}{c^2}A - \rho B. \tag{2.28}$$

Using one-sided second order finite differences, for the normal derivative, we obtain in the first octant:

$$\begin{aligned}&\frac{n_x}{2\Delta x}(-\rho_{i+2,j,k}+4\rho_{i+1,j,k}-3\rho_{i,j,k})\\+&\frac{n_y}{2\Delta y}(-\rho_{i,j+2,k}+4\rho_{i,j+1,k}-3\rho_{i,j,k})\\+&\frac{n_z}{2\Delta z}(-\rho_{i,j,k+2}+4\rho_{i,j,k+1}-3\rho_{i,j,k})+\rho_{i,j,k}B_{i,j,k}=\frac{1}{c^2}A_{i,j,k}\end{aligned} \tag{2.29}$$

If we rearrange this, to solve for ρ we have our desired update rule for ρ:

$$\begin{aligned}\rho_{i,j,k}=\frac{2\Delta x\Delta y\Delta z}{c^2 div}A_{i,j,k}&-\frac{n_x\Delta y\Delta z}{div}(-\rho_{i+2,j,k}+4\rho_{i+1,j,k})\\&-\frac{n_y\Delta x\Delta z}{div}(-\rho_{i,j+2,k}+4\rho_{i,j+1,k})\\&-\frac{n_z\Delta x\Delta y}{div}(-\rho_{i,j,k+2}+4\rho_{i,j,k+1}).\end{aligned} \tag{2.30}$$

with $div=-3n_x\Delta y\Delta z-3n_y\Delta x\Delta z-3n_z\Delta x\Delta y+2\Delta x\Delta y\Delta z B_{i,j,k}$. The quantities A and B can be calculated beforehand, since the inner fluid points have already been updated.

Evidently, this also imposes some restrictions on the feasible geometry: We need to ensure that every fluid domain is at least three cells wide, and no obstacle cells, with fluid on opposite sides, exist. In practice, we can achieve this by refining the underlying grid, but due the cubic scaling of the corresponding number of cells and the grids uniform nature, this can increase computing time by a critical margin.

2.5 Stability

As usual with explicit methods, there is some sort of mandatory Courant-Friedrichs-Levy condition (CFL), that needs to be fulfilled to get any form of stability for our scheme. In the compressible setting, it is important to remember the fact, that we not only need to look at the flow speeds itself, but also have to consider the speed of sound, which is the rate at which pressure waves propagate through our domain. So the very thing, that allows us to use an explicit method in the first place, i.e. a finite propagation speed, also sets the limits in how far we can go within a single time step. So despite our demand to stay below a certain Mach number, we face the problem, that very low number makes our code inefficient, due to the faced restriction on time steps.

Because of the complexity of the underlying equations, we cannot hope to get a rigorous stability requirement in closed-form for our technique, but the

following semi-empirical criterion given by Tannehill [THR76] seems to be relatively reliable in giving us a good estimate

$$\Delta t \leq \frac{\tau}{1+2/Re_\Delta}\left[\frac{|u|}{\Delta x}+\frac{|v|}{\Delta y}+\frac{|w|}{\Delta z}+v_{sound}\sqrt{\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}+\frac{1}{\Delta z^2}}\right]^{-1}, \tag{2.31}$$

where $Re_\Delta = min(\rho|u|\Delta x/\mu, \rho|v|\Delta y/\mu, \rho|w|\Delta z/\mu)$ and $\tau \approx 0.8$ is a safety factor. This criterion gives us a local restriction for each cell, so we have to minimize this quantity over our mesh, subsequently, to get a globally admissible time step.

The term on the right-hand side

$$(\Delta t)_{CFL} \leq \left[\frac{|u|}{\Delta x}+\frac{|v|}{\Delta y}+\frac{|w|}{\Delta z}+v_{sound}\sqrt{\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}+\frac{1}{\Delta z^2}}\right]^{-1}, \tag{2.32}$$

was identified by MacCormack (see [Mac71]) as the inviscid CFL condition. He obtained this criterium, by studying the amplification of Fourier components of the linearised equation.
For a cubic grid and M small enough this reduces to the well-known, much simpler condition

$$\Delta t \leq \frac{1}{2}\frac{\Delta x}{v_{sound}}. \tag{2.33}$$

Remember though, that these can be necessary conditions, at best, which are mostly derived from linearised equations and don't take difficult boundary regions into account. Hence, in practice, finding a suitable time step still comes with some difficulties and we will see, that either way, this imposes some restrictions on the problems we can solve. We will discuss this later in detail for some of the examples in Chapter 5.

2.6 Parallelization

Even with the most modern CPUs, clocking in the multiple GHz region, tackling interesting fluid problems is only possible, if we employ many of them at the same time. In our case, we want to use a big cluster of CPUs later on. In order to do this, it is necessary to split our problem up and give each processor a piece of the computation, that can, for the most part, be solved independently. As most CFD codes, we arrange this, by dividing our domain spatially into chunks of roughly the same computational complexity and then assigning each of these sub-domains to its own processor. In this process, a big advantage of having a rectangular grid comes into play: Dividing the computational cost evenly among the processors is easy, since the grid

points of our mesh are distributed evenly among our domain, so we can assume that dividing our domain spatially into blocks of the same size is also computationally efficient. Since cubes posses the best ratio of volume to surface, depending on the shape of our domain we try to avoid large aspect ratios in our sub-domains. In this manner, the boundaries between adjacent processors are split into six distinct directions and communication between the processors can be handled relatively easy: Our update scheme requires the values of fluid variables on neighbouring points. If we are close to the border of our sub-domain, these points might belong to another processor and must therefore be communicated in some way, since each processor is fitted with its own chunk of memory. Even on shared memory machines, arrangements have to be made, in order to prevent simultaneous access to the same memory blocks. In practice, we exchange these border values after each time step and store them in so-called ghost cells, that surround our actual domain and serve just this purpose of having a local copy of the values on neighbouring cells. This way we can handle all interfacing between the processors in one go, and afterwards, each processor can run the update scheme for its own variables independently of the rest.

In the actual code, this communication is handled with the Message Passing Interface (MPI). For the present work, we rely on parallelization routines already existing in the NaSt3DGP fluid solver. For further details on the exact routines, we refer to [GDN98]. If we take a look back at our update scheme, we notice that we used up to three cell neighbours in a given direction in our density boundary condition, so we will also need a minimum of three layers of ghost cells to lose no information. Additionally, at some point, we access one neighbouring point in diagonal directions, so we have to take care, that one layer of corner ghost cells is exchanged, as well. Figure 2.1 shows a schematic overview of the required communication in 2D. Despite this seeming bulk of information that needs to be transferred between processors, we expect a pretty decent efficiency in our parallelization. The results of some practical measurements on this question can be found in Section 5.1.

2.7 An overview of the full algorithm

In this section, we want to give a short overview of the implemented algorithm. Again, the algorithm proceeds in a totally linear update routine, in the course of which no system of equations has to be solved. For the momentum variables, as well as the density, we need two separate fields for the full time step and the predicted values, respectively, since the information of either values is needed to update the other. In this context, all variables representing predicted values, will be indicated by a $*$. Since we don't use the velocity components for updating the inner grid cells, we can store both predicted and corrected

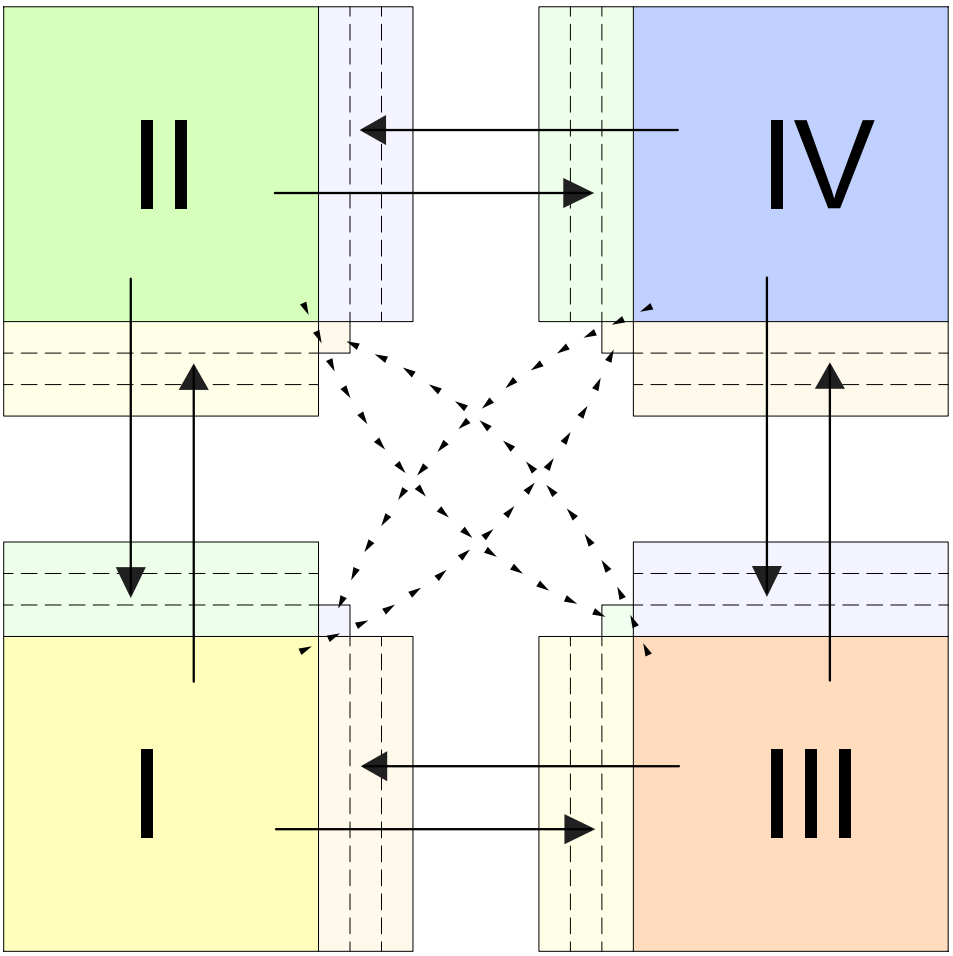

Figure 2.1: Schematic overview of the required communication between four neighbouring processors in 2D, using tree layers of ghost cells and one corner cell

values in the same variables, in this case. In Algorithm 2.1 a schematic overview of the full algorithm is presented.

Algorithm 2.1 Schematic overview of the main loop of our program

```
repeat
    n++;                                          ▷ Augment time step
    Timestep();                                   ▷ Calculate new time step
    mode = n mod 8                                ▷ Calculate cyclic scheme

    PredictorStep(mode);                          ▷ Predictor step
    CalculateUVWStar();
    BCPredictorStep();

    Par->CommMatrix(RhoStar);                     ▷ Communication for * variables
    Par->CommMatrix(RhoUStar);
    Par->CommMatrix(RhoVStar);
    Par->CommMatrix(RhoWStar);
    Par->CommMatrix(U);
    Par->CommMatrix(V);
    Par->CommMatrix(W);

    CorrectorStep( mode );                        ▷ Corrector step
    CalculateUVW();
    BCCorrectorStep();

    Par->CommMatrix(Rho);                         ▷ Comunication for n+1 variables.
    Par->CommMatrix(RhoU);
    Par->CommMatrix(RhoV);
    Par->CommMatrix(RhoW);
    Par->CommMatrix(U);
    Par->CommMatrix(V);
    Par->CommMatrix(W);

    Output();                                     ▷ Output data
until n == nMax || t >= tMax
```

The main part of our program is structured by a do-while loop, representing consecutive time steps of our algorithm. We begin each given step, by augmenting the respective count n by one and calculating the new time step Δt for the upcoming update, according to (2.31). Next, we set the mode for the current differencing scheme according to Table 2.1, cycling through possibilities 1 to 8. The rest of the algorithm is divided into two blocks, that resemble each other, except for the slightly different update routines

discussed in Section 2.3: First we update the inner fluid points in the main predictor/corrector step, subsequently, we calculate our velocities by a simple division. Finally, we use the gathered information, to provide boundary values for the next iteration and communicate the needed values to the neighbouring ghost cells. At the end of each time step we output our data, if so desired. Since the existing NaSt3DGP output routines are designed for the staggered grid setup, we adopted our own version of them, giving us straight outputs of our fluid variables in appropriate VTK (Visualization Toolkit) file formats.

Chapter 3

Geometry representation

What we have seen thus far, is basically a full solver of the compressible Navier-Stokes equations in a channel, with different boundary conditions for the walls, the only restriction being the demand of staying in the low Mach number regime. But besides being able to solve some simple benchmarks, we want to have a method to represent more complex geometries, that allow us to tackle more interesting problems, that are related to real-world applications. In order to do this, some different methods can be considered. A simple flag-field is easy to implement, but also lacks the kind of flexibility, we seek in the long run. Another problem with this approach is the bad approximation of the surface normal of our geometry. Another approach is the so-called level set method (LSM) which we want to explore in the upcoming section of this work.

3.1 The level set method

The level set method was first outlined by Osher and Sethian in their 1988 paper [OS88]. It is a so-called front tracking method, that very generally allows one to capture the interface between two distinct regions of a domain. Today it is used in many areas such as image processing, computer graphics or, likewise, computational fluid dynamics (for a more extensive overview see for example [OF03]). The basic idea is to have a function φ, which is defined on the higher dimensional surrounding space, that encodes the movement of the evolving surface. What seems at first to be a not very effective way of dealing with the involved information, turns out to give a very high versatility, when it comes to topological changes of the geometry. In contrast to some other methods, the surface never has to be parametrized and is only modified implicitly, by adjusting the function φ. Things like an object splitting apart or the merging of objects, are handled naturally by this framework, which makes it very robust. Let us see how this works in a bit more detail.

We keep our notation close to Losasso, Fedkiw and Osher ([LFO06]), who give a neat introduction to level set methods. If we call the enclosed region of space Ω^- and the interface between Ω^- and it's complement Γ, we demand our level set function φ to be a Lipschitz continuos function with the following characterizing properties:

$$\varphi(\boldsymbol{x}, t) \leq 0 \text{ for } x \in \Omega^- \tag{3.1}$$

$$\varphi(\boldsymbol{x}, t) > 0 \text{ for } x \notin \Omega^- \tag{3.2}$$

Then clearly, we have

$$\Gamma(t) = \{\boldsymbol{x} | \varphi(\boldsymbol{x}, t) = 0\}. \tag{3.3}$$

We can now implement an evolution of this interface, by a very general velocity field $\boldsymbol{a}(\boldsymbol{x}, t)$, that transports the interface. We call this field $\boldsymbol{a}$ to distinguish it from the velocity $\boldsymbol{u}$, of our flow, since the two need not coincide, here. For this purpose, the velocity needs only to be given in a neighbourhood of the interface. The equation that describes the advection of the level set is then given by

$$\partial_t \varphi + \boldsymbol{a} \cdot \nabla \varphi = 0. \tag{3.4}$$

It was already introduced in the initial paper by Osher and Sethian ([OS88]) and is just called the *level set equation*. So far, we still have a lot of freedom of extending φ ouside the zero level set. Since, for numerical accuracy, we want the function to be as smooth as possible, we prefer it to be a signed distance function, at least in a certain band around the interface. Unfortunately this property is not preserved by the solutions to the above transport equation. While small deviations don't cause much harm, over time we need a method to restore this property, given our current state of the interface. There are some different algorithms around to achieve this *reinitialization*, the details of which we don't want to mention here (see for example [OF01]).

One of the key benefits of the technique, which we want to exploit especially, is the very natural and yet very accurate access to geometric information like the normal and curvature of our geometry. The normal to our interface Γ is simply given by

$$\boldsymbol{n}(\boldsymbol{x}, t) = \frac{\nabla \varphi(\boldsymbol{x}, t)}{|\nabla \varphi(\boldsymbol{x}, t)|}, \tag{3.5}$$

whereas the mean curvature κ can subsequently be determined as the divergence of the former

$$\kappa(\boldsymbol{x}, t) = \nabla \cdot \left(\frac{\nabla \varphi(\boldsymbol{x}, t)}{|\nabla \varphi(\boldsymbol{x}, t)|} \right). \tag{3.6}$$

The most often mentioned problem with the level set method is that the discretisation of the level set equation can lead to dissipation, sometimes resulting in a severe mass loss. Many improvements have been proposed on this front, the most promising of which is perhaps a combination with Lagrangian particle tracking [Enr+02].

Level set in NaSt In the NaSt3DGP code, the level set method is already in use, among other applications, to handle the interface in two-phase flow problems and simulate free surface flows. The transport equation (3.4) is discretised with a fifth order WENO scheme, which prevents edges and other geometric discontinuities, that can arise at the merging or splitting of two objects, from leading to numerical instabilities, which in turn ensures the unproblematic handling of such topological singularities. The reinitialization is handled by a method first appearing in [SSO94]. The idea is to compute the steady-state solution to the following PDE of Hamilton-Jacobi type

$$d_t + \operatorname{sign}(\varphi)\,(|\nabla d| - 1) = 0, \tag{3.7}$$

where sign is the standard signum function and d is initialized to our current level set function φ. Afterwards we replace φ with d. The steady-state solution of this equation, is the wanted distance function, while the zero level set is left untouched. The spatial discretisation of the equation, is once again handled by the fifth order WENO scheme, leading to much better results in terms of mass conservation, than comparable schemes of lower order, while the time discretisation is done via a third order Runge-Kutta treatment. Additionally, the appearing signum function is smoothed out, leading to further improvements. Since conservation of mass can still be problematic in certain circumstances an optional Picard fixed point iteration is in place to retouch the resulting level set function afterwards. For the more involved details concerning these procedures, we refer to the dissertation of Croce [Cro10].

3.2 Geometry import via the stl file format

In this section, we want to briefly explain how we set up our level set function in the first place, starting for example from a given CAD model. The stl file format originally stems from the software *Stereolithography*. Nowadays it is a very common format for exchanging geometry in a tessellated form and can be considered an industry standard, especially in the growing field of rapid prototyping. Despite some shortcomings, it's widespread use was the primary reason, we chose stl as a way to transfer our geometry, but we also considered it for its simplicity. The idea of stl, is basically to have a list of triangles, that represent the surface of your geometry. The outside normal of the object is determined by the right-hand rule. There is both an ASCII

and a binary version of the stl file format. Due to the much improved file sizes and even simpler structure, we only considered the binary version for our code.

Constructing the level set function

Once the stl-file is loaded, a point in polygon test is used to implement a routine *IsInside*, which decides if a given grid point, is inside the object spanned by the stl surface or outside. This is done via a so-called ray casting algorithm. First, a direction is determined, in which an outgoing ray strikes no vertex or edge of the triangles, that make up our model. Then, for each of the triangles, we decide if the ray passes through it. Therefore, we initially check, if it intersects the plane spanned by that triangle, and if so determine if the intersection is inside the triangle. Counting up the number of triangles, that our ray hits on its way, we get the result: If the number is odd, we are located in the region enclosed by the polygon surface, otherwise we are outside. With the use of this function, a characteristic function of the geometry is created, and the corresponding flag field can already be put in place. The level set function, finally, is created from this characteristic function by applying the reinitialization procedure, described earlier, to it. By this procedure a full level set distance function is generated on the grid, and can be kept until the end of the simulation, since at the current stage of our code, we don't manipulate the function itself, but only translate and/or rotate it.

3.3 Comparison to the flag-field

To demonstrate the advantage of the level set method over the pure flag-field, we want to exemplarily show the three different stages of an imported model in our code. The chosen example is the model of a motorcycle helmet attached to a very simple torso. The original stl file consists of ca. 3.500 triangles and was modelled with the CAD-software Pro/ENGINEER. It is positioned in a cubic block of side length 0.6 m, which is discretised with a grid consisting of 60 cells in each direction, which corresponds to 216.000 cells in total. The three variants, visualized with ParaView, are shown in Figures 3.1 to 3.3, according to their order of appearance during a run of the algorithm. As one can clearly see, at such a resolution,the general shape of the model is represented relatively accurate by the flag-field, but the surface normal being restricted to 6 different directions, makes the approximation still look very clunky. In comparison, the level set contour looks very smooth, one can easily see the much improved behaviour, regarding normal directions, which we extensively use in our boundary conditions, as well as in the computation of forces on our obstacles, as we will see in the next chapter. The only drawback being, that sharp edges also get smoothed out in the process and have a small radius applied to them.

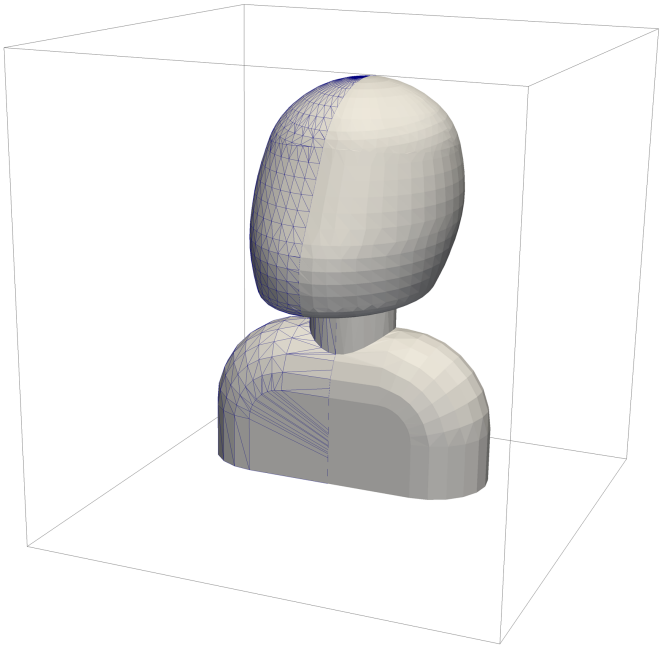

Figure 3.1: Step 1: Stl model of a motorcycle helmet, left half visualized with edges

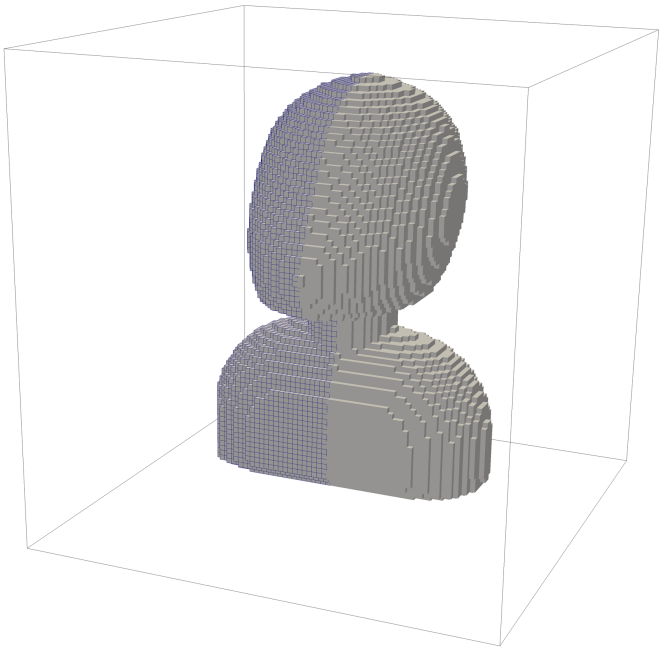

Figure 3.2: Step 2: Flag field threshold of a motorcycle helmet, left half visualized with edges

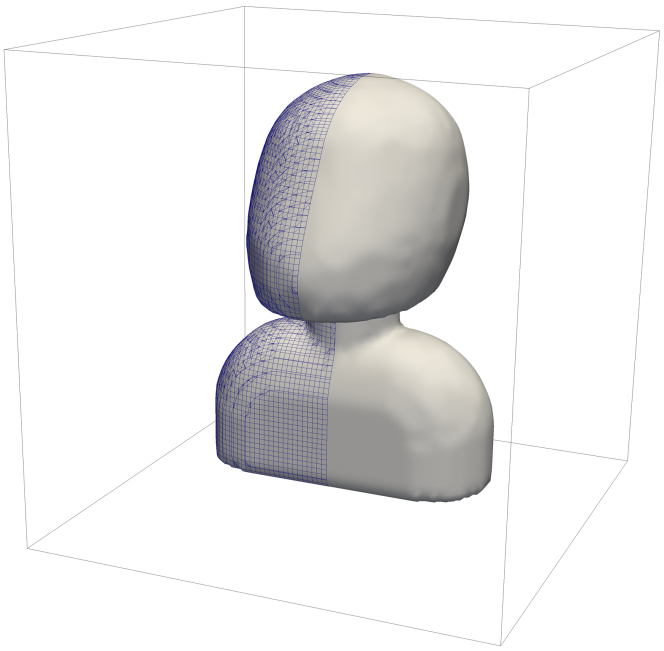

Figure 3.3: Step 3: Level set contour of a motorcycle helmet, left half visualized with edges

Chapter 4

Simulation of moving objects

Now, that we have static geometries implemented, we want to look at the possibility of simulating geometries, evolving over time. The easiest class of such problems is clearly a constant object moving through the scene. We want to tackle this problem in this section, with the help of the already implemented level set method. In order to go ahead, we need to make sure, that our boundary conditions still work on a moving boundary.

4.1 Adjusting the boundary conditions

First of all, we assume that all the boundary points on the object are only of no-slip type. Although a generalization to allow inflow/outflow boundaries could easily be implemented, we don't want to consider this case here and most applications don't require it, anyway. Let us remember the equation for such no-slip boundaries, again only accounting for the first octant:

$$\begin{aligned} u_{i,j,k} = \frac{1}{c}\Big[& -\Delta\hat{x}\Delta y\Delta z(-u_{i+2,j,k} + 4u_{i+1,j,k}) \\ & -\Delta x\Delta\hat{y}\Delta z(-u_{i,j+2,k} + 4u_{i,j+1,k}) \\ & -\Delta x\Delta y\Delta\hat{z}(-u_{i,j,k+2} + 4u_{i,j,k+1}) + 2\Delta x\Delta y\Delta z(u_{sur})\Big], \end{aligned} \tag{4.1}$$

with $c = 2\Delta x\Delta y\Delta z - 3\left[\Delta\hat{x}\Delta y\Delta z + \Delta x\Delta\hat{y}\Delta z + \Delta x\Delta y\Delta\hat{z}\right]$.

The only thing we need to do now, is to feed the velocity of the objects movement (represented by its centre of mass) into this equation as the value for u_{sur}, v_{sur} or w_{sur}, accordingly. The equation itself works right out of the box.

Velocity prediction for leaking boundary points

Looking closer, there is one other small thing we need to consider, though. Our object will be moved after the boundary values for the corrector step are set, and right before we advance to the next time step, starting with the usual predictor step. So there usually will be points that were inside the obstacle before, and now become the new boundary. Both, velocity and density values at these points will be accessed in the following predictor step, when updating their neighbours, that in turn were part of the boundary and now changed to be regular fluid cells. In other words, points "leaking" from the geometry need to be addressed. We need to find a way to predict a sensible velocity and density for them. This will, once again, be done by applying the one-sided second order differences from (2.22) to a taylor expansion. Assuming we are in the first octant, we have

$$u_{i,j,k} \approx u_{i+1,j,k} - \Delta x (u_x)_{i+1,j,k}. \tag{4.2}$$

Inserting our differencing scheme from (2.22), we get

$$u_{i,j,k} \approx \frac{1}{2}(5u_{i+1,j,k} - 4u_{i+2,j,k} + u_{i+3,j,k}). \tag{4.3}$$

Since we can do the analogous approximation in directions y and z, we average them with the normal $\boldsymbol{n}$ as a weight, to get our final update rule:

$$\begin{aligned} u_{i,j,k} = &\frac{n_x^2}{2}(5u_{i+1,j,k} - 4u_{i+2,j,k} + u_{i+3,j,k}) \\ &+ \frac{n_y^2}{2}(5u_{i,j+1,k} - 4u_{i,j+2,k} + u_{i,j+3,k}) \\ &+ \frac{n_z^2}{2}(5u_{i,j,k+1} - 4u_{i,j,k+2} + u_{i,j,k+3}) \end{aligned} \tag{4.4}$$

Exactly the same reasoning can be used for the density, as well. As a final note, we remember, that we already use three layers of ghost cells in the parallelization, so no adaptations have to be carried out on this front.

Tracking the boundary

In the last section we have discussed the changes to our boundary conditions, that need to be implemented if we want to simulate moving obstacles. In our code, boundary points are saved in lists, gathering points of the same boundary type, which are established on the basis of our flag field. In the end, moving our geometry is realized by moving the level set function, by which it is defined. Hence, in order to keep track of our boundary points, we

need to update our flag field, as well as the corresponding lists, accordingly. In Practice, after each time step, we go through our level set field and check for sign changes among neighbouring points. If we detect a changing sign we set the appropriate flag. If a given cell, was marked as an obstacle cell before, and is now on the boundary, we additionally mark it as *leaking*. This way, we mark it to receive the special care discussed in the last subsection. After this procedure, we rebuild our lists and update our leaking points, as described.

4.2 Moving geometries with a prescribed velocity

Fundamentally, we are interested in having both, objects with a prescribed velocity, as well as geometries driven by the flow. In Chapter 3 we described, how to generally transport a level set function by a general velocity field $\boldsymbol{a}$. While this setting gives us a very high flexibility, in terms of modifying our object, in the course of this thesis, we want to restrict ourselves to the special case of globally transporting the level set, i.e. moving our object as a whole, rather than locally moving its boundaries, and therefore possibly altering its shape, while the latter still remains as a future option. Let $\boldsymbol{u}_{CoM}(t)$ be the velocity we want to assign to our object, which is represented by its centre of mass (CoM). Then, in the general setting from above, our case can be described as

$$\boldsymbol{a}(\boldsymbol{x}, t) = \boldsymbol{u}_{CoM}(t). \tag{4.5}$$

Let $\boldsymbol{s} = (s_x, s_y, s_z)$ bet the distance our obstacle moves in a given time step. Then we evidently have

$$\begin{aligned} s_x(t_n) &= u_{CoM}(t_n)\,\Delta t \\ s_y(t_n) &= v_{CoM}(t_n)\,\Delta t \\ s_z(t_n) &= w_{CoM}(t_n)\,\Delta t \end{aligned} \tag{4.6}$$

We use a Semi-Lagrangian ansatz to implement this movement, which means, that we calculate new values at our grid points, by going backwards in time to deduce their initial position before the movement occurred. Let us look at the situation for the grid point (i,j,k). Before the translation it was located at coordinates

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} i\Delta x - s_x(t_n) \\ j\Delta y - s_y(t_n) \\ k\Delta z - s_z(t_n) \end{pmatrix} \tag{4.7}$$

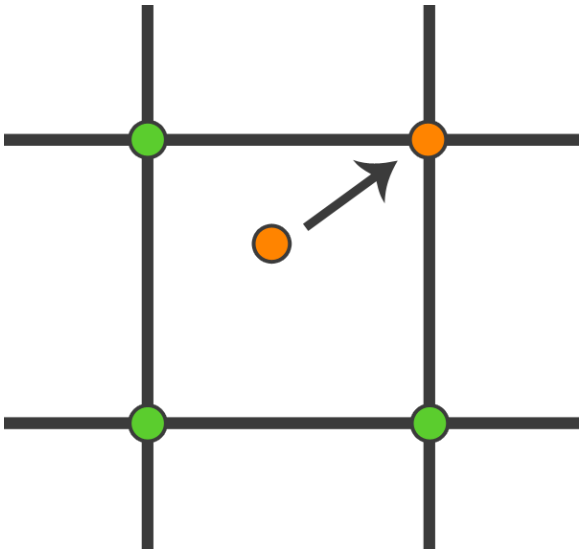

Figure 4.1: Basic idea of the Semi-Lagrangian ansatz

Let $p_1 = (i_1, j_1, k_1)$ and $p_2 = (i_2, j_2, k_2)$ be the two unique grid points with coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2) such that

$$\begin{aligned} x_1 < x < x_2 = x_1 + \Delta x \\ y_1 < y < y_2 = y_1 + \Delta y \\ z_1 < z < z_2 = z_1 + \Delta z \end{aligned} \tag{4.8}$$

We interpolate the new value for the level set function L at (i,j,k) between the eight grid points surrounding its original position:

$$\begin{aligned} L_{i,j,k}(t_{n+1}) = \frac{1}{\Delta x \Delta y \Delta z} \Big[& (x_2 - x)(y_2 - y)(z_2 - z) L_{i_1,j_1,k_1}(t_n) \\ & + (x - x_1)(y_2 - y)(z_2 - z) L_{i_2,j_1,k_1}(t_n) \\ & + (x_2 - x)(y - y_1)(z_2 - z) L_{i_1,j_2,k_1}(t_n) \\ & + (x - x_1)(y - y_1)(z_2 - z) L_{i_2,j_2,k_1}(t_n) \\ & + (x_2 - x)(y_2 - y)(z - z_1) L_{i_1,j_1,k_2}(t_n) \\ & + (x - x_1)(y_2 - y)(z - z_1) L_{i_2,j_1,k_2}(t_n) \\ & + (x_2 - x)(y - y_1)(z - z_1) L_{i_1,j_2,k_2}(t_n) \\ & + (x - x_1)(y - y_1)(z - z_1) L_{i_2,j_2,k_2}(t_n) \Big] \end{aligned} \tag{4.9}$$

Obviously, we need to make sure, that our object moves less than one cell length per time step with this scheme, but this is automatically secured by our stability condition including the speed of sound, since we don't want to simulate objects anywhere close to the speed of sound.

4.3 Translation vs. rotation

Up until this point, we have only talked about translational movement, but in principle, the above method could be easily extended to rotational movement, as well. The only thing that needed to be done is to incorporate the rotation into the Semi-Lagrangian ansatz, where we compute the original position of our point. There is another problem with this method altogether, though, which we want to explain in this Section, and which finally led to the decision, to only consider translations for the present thesis.

When trying the above method in practice, we soon encountered terrible problems with the stability of our level set function. Even after very short times, we could see a severe mass loss and some deformations of the object, as well. Apparently, interpolating the level set function in such a manner doesn't work very well, if you do it many times over. This effect is especially grave in our case, due to the small time steps mandated by the CFL-condition, resulting in a huge number of successive interpolations. Our first idea to overcome this problem, was to not interpolate successively, for each time step, but to always start from the original level set function, that was generated by the reinitialization procedure. For this to work, we only have to add up the distances from Equation 4.6 as we go. This way, there's never more than one interpolation, since we always start fresh with the original level-set field. In turn, however, the position of our point of interest and its original location, were we interpolate the new value, drift apart, in the same manner as we move our object. This is no problem in a serial environment, where only one processor is involved, and consequently has access to the complete level set field. If we want to run our algorithm in a parallel fashion on many processors, though, we face the issue of only having a local piece of the field in the memory of each processor. Both, having to communicate all values where cells of a given processor emerged from, which is a very elaborate and tedious procedure, as well as having a global level set function for each processor of its own, which is blatantly inefficient memory-wise and defeats the purpose of the parallelization in the first place, are no practically useful solutions to this.

For translations, at least, there is a more elegant solution: We can do the above, i.e. interpolating from the initial level set function, until our object has moved by a whole grid cell in any given direction. When this happens, we move our original level set, which we use as a source for the interpolation, by one grid cell in this direction. This process doesn't require any interpolation, since the grid is mapped to itself. This way, the origin of each grid pont always keeps track with the movement of the obstacle, and therefore no communication is needed, yet still, there is never more than a single interpolation required to deduce the value for a given position, so

we don't run into the problems discussed before. The only downside to this method is, that there is no obvious generalization to rotations, since there is no natural analogue of moving by one grid cell. The first angle at which a rotation of a (quadratic) grid aligns with its initial origin is at 90°, which is way too coarse to help with the problem at hand. Since no other solution was found in the time-frame of this work, we restrict ourselves to translational movement for the point being, although the rest of the theory could easily be adapted, to the general case. In the end everything comes down to a parallelization issue, which seems difficult to solve, but there is no point running in serial, whatsoever.

4.4 Bidirectional coupling

In the following Section, we want to see how we can generalize the above to allow for bidirectional coupling, too. Again, we restrict ourselves to translational movements, although the coupling would work exactly the same for rotations. In some way, this bidirectional coupling has to be based on computing the forces our fluid exercises on the obstacle at hand. Once we have done this, we just feed these forces back into the loop by adjusting our Centre-of-Mass velocity $\boldsymbol{u}_{CoM}$ via Newton's law. In the end our problem basically comes down to computing drag/lift forces on a body, as well as its mass or rather its volume. The calculation of both of these quantities relies on integration. Most generally, we can express the force on an obstacle immersed into our fluid, by integrating the stress among its surface, in other words

$$\boldsymbol{F} = \int_S \boldsymbol{\sigma} \cdot \boldsymbol{n} \, \mathrm{d}S, \tag{4.10}$$

where $\boldsymbol{n}$ is of course the outer normal to the surface S. Now, to be consistent with the equations presented in Chapter 2, our total stress $\boldsymbol{\sigma}$ is given by

$$\boldsymbol{\sigma} = -p\,\mathbb{I} + \mu(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T) - \lambda(\nabla \cdot \boldsymbol{u})\,\mathbb{I}, \tag{4.11}$$

where we discretise all appearing derivatives with central differences. Inserting this above, all that's left to do, is to approximate the integral in our discrete setting. There are many different ways of doing this for a fixed mesh, some using very elaborate interpolation techniques. In 2012 Haeri and Shrimpton published a review paper on a few of these methods ([HS12]). Here, we want to keep things a little simpler, though. Again, we make use of the normal, which is readily provided by our level-set function. To discretise the integral itself, first, we want to bring it to a slightly different form. Since $S = \{\boldsymbol{x} | \varphi(\boldsymbol{x}) = 0\}$ and $|\nabla \varphi| = 1$, using a change of variables, we can rewrite

(4.10) as follows

$$\boldsymbol{F} = \int_S \boldsymbol{\sigma} \cdot \boldsymbol{n} \, \mathrm{d}S = \int_\Omega \boldsymbol{\sigma} \cdot \boldsymbol{n} \, \delta(\varphi(\boldsymbol{x})) \, \mathrm{d}\boldsymbol{x}, \tag{4.12}$$

where δ is the Dirac delta distribution. Rigorously, this follows easily from the coarea formula. Using a smooth standard approximation δ_h of finite width h to the delta distribution, we can now approximate our integral discretely

$$\boldsymbol{F} \approx \sum_{i,j,k} (\boldsymbol{\sigma}(i,j,k) \cdot \boldsymbol{n}) \, \delta_h(L(i,j,k)) \, d_x(i) d_y(j) d_z(k) \tag{4.13}$$

In practice, a thickness of 1.75 cell diameters was used. Since this process smears out the delta function in both directions, we have to make sure, that we don't use any points inside our obstacle, where no information about velocity and pressure exists. Therefore, we make one final change to our actual approach:

$$\boldsymbol{F} \approx \sum_{i,j,k} (\boldsymbol{\sigma}(i,j,k) \cdot \boldsymbol{n}) \, \delta_h(L(i,j,k) - 0.5h) \, d_x(i) d_y(j) d_z(k) \tag{4.14}$$

Figuratively, we push out our integration surface by half of this width, which doesn't alter the value of the corresponding integral. In fact, due to the conservation of momentum, any closed surface containing our obstacle should theoretically give the correct value. Finally, for the calculation of the volume, all we have to do is basically count the cells inside our obstacle, the details of which will be omitted, here.

Chapter 5

Example results

In this chapter we want to show a selection of results, we could achieve on the basis of the previously presented theory. Each computation is performed to show a specific feature of the code and is only exemplary. A multitude of different possibilities exist beyond the presented scenarios. In general, the chapter is divided roughly into two areas: The first two sections are devoted to validate our code in different ways. Subsequently, we want to explore the possibilities that can be realized with the algorithms at hand, showing some nice examples, in the progress. All visualizations throughout the chapter have been carried out using ParaView and the Visualization Toolkit (VTK). The label *vectors Magnitude* in the corresponding figures refers to the magnitude of the velocity field $\boldsymbol{u}$, always expressed in m/s. Allthough our theory, as well as our code is formulated to operate with density, rather than pressure, for the output and analysis of the flow, we decided to opt for a zero-centred pressure, which is simply given by

$$p = v_{sound}^2(\rho - \rho_0), \tag{5.1}$$

where ρ_0 is the resting density of the fluid. The two convey the same information, yet the visualization in terms of pressure is easier to present and more familiar, coming from the incompressible case. All appearing plots were done with the aid of Gnuplot.

5.1 First Tests

Having implemented the preceding methods, we want to test them with some simple cases, as a preliminary point, before going over to more complex problems. Therefore, in this section, we try to replicate some standard driven cavity results, look at the progression of L_2 errors on different grid resolutions and finally examine the parallel scaling of our code on a CPU cluster, to get an idea how it performs in the most general setting.

Driven cavity

As a first test, we try to replicate the results from Perrin and Hou ([PH06]) for a driven cavity scenario in 2D. To recreate their setting, we use the symmetric boundary conditions presented in Section 2.4 to create a pseudo 2D form of the algorithm. Physically, the two experiments are equivalent, but a confirmation of the results still gives us a first hint at the correctness of our code, as well as the underlying equations.

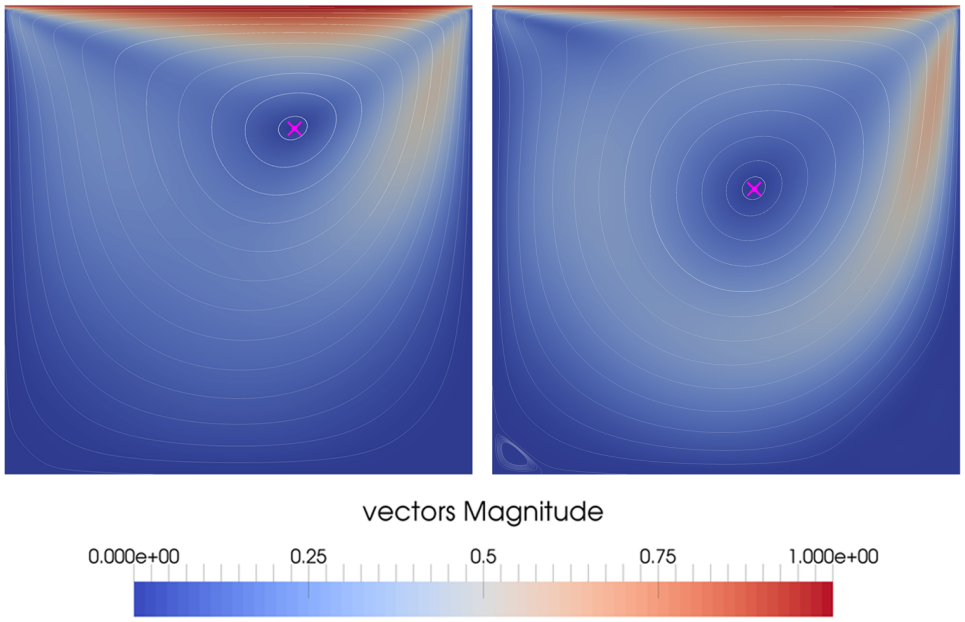

Figure 5.1: Velocity field of a driven cavity on a 256x256 grid at $Re = 100$ (left) and $Re = 400$ (right) and $U = 1m/s$; stopped after simulated time of 24 s. The purple crosses mark the vortex centres found by [Hou+95].

The calculations are performed on a uniform grid of 256x256 cells, representing the domain $\Omega = [0, 1] \times [0, 1]$, and 32 cells in the (symmetric) z-direction. The flow is driven by the upper border of the domain moving at a speed of 1 m/s to the right. No-slip conditions are enforced on all other walls of the cavity. To remain in the nearly incompressible region the speed of sound is set to 10 m/s, resulting in a Mach number $M = 0.1$. The experiment is then conducted at a Reynolds number of 100 and 400, each, and run until steady state is reached.The resulting velocity fields and corresponding streamlines are shown in Figure 5.1. In both cases a large primary eddy vortex can be found spanning most of the cavity, withsome smaller vortices emerging in the bottom corners of the domain. In accordance with our expectations, in the

course of the higher Reynolds number, the primary vortex moves closer to the middle of the cavity and the secondary vortices become more distinct, due to the lower viscosity. The centres of the primary vortices are in good accordance with the results found by Ghia et al. ([GGS82]) and Hou et al. ([Hou+95]), who give coordinates of $(0.6196, 0.7373)$ at $Re = 100$ and $(0.5608, 0.6078)$ at $Re = 400$ using a Lattice-Boltzmann method. Eventually, with higher Reynolds numbers, the vortices will become unstable as reported by Ghia et al. All in all we can securely replicate the results found in the literature and at first glance computing times seem to be competitive, as well.

Progression of L_2-Errors

Next, we want to perform further tests on the accuracy of our solutions, especially when increasing grid resolutions. Due to the lack of exact solutions for the compressible Navier-Stokes equations, showing any form of convergence proves very difficult. What we can do instead, to estimate convergence order, is to compute a solution on a very fine mesh and examine the behaviour of the algorithm when approximating it with coarser variants from below. We want to use the experiment above, and calculate the L_2 error for different resolutions compared to a reference grid of 800x800 cells. Grid sizes of the related coarse meshes varied from 25x25 cells up to 400x400 cells, with a two-fold margin left over to our reference solution, in order for the assumed negligible error on the fine grid not to distort the picture. To capture possible time-dependant effects, the simulation wasn't run until steady-state, but was stopped after a fixed time of 4.0 s. Throughout the study, the Reynolds number was set to 100. For the comparison, the result on the coarser grid was then interpolated to the finer reference grid, and an approximation to the L_2 norm was calculated in the following way:

$$\begin{aligned} \|U\|_{L_2} &= \left(\int_\Omega |U(x,y,z)|^2 \, \mathrm{d}\mathcal{L}^3(x,y,z) \right)^{1/2} \\ &\approx \left(\sum_{i,j,k} |U(i,j,k)|^2 d_x(i) d_y(j) d_z(k) \right)^{1/2} \end{aligned} \tag{5.2}$$

Having established the norm, the absolute error ϵ and relative error η are then defined as follows:

$$\epsilon_{U,L_2} = \|U_{fine} - U_{coarse}\|_{L_2} \tag{5.3}$$

$$\eta_{U,L_2} = \frac{\|U_{fine} - U_{coarse}\|_{L_2}}{\|U_{fine}\|_{L_2}} \tag{5.4}$$

In practice, the filters *ResampleWithDataset*, *IntegrateVariables* and *Calculator*, provided by the ParaView Software, were used to perform the required computations. In the same manner we can calculate the related errors for the pressure. The results of the study are listed in Table 5.1 alongside a logarithmic plot, comparing the data with an appropriate fit in Figure 5.2, to estimate the order of convergence. In conclusion, the results are pretty encouraging: The order of convergence, for both the velocity and the pressure increases, the higher our resolution goes. In general, the behaviour for the velocity, going from 1.53 in the beginning, to a value of 2.33 in the final step, seems to be slightly better than for the pressure. The last two values, even going above the proposed second order, are probably due to the fact, that our assumption of a negligible error on the reference grid, becomes less accurate, the closer we get to it. Therefore, especially the last entry, for a resolution of 400 points, is probably overshooting the actual value. The corresponding fit, which estimates an order of 1.79 for the velocity and around 1.46 for the pressure, should give us a more accurate picture of our study. Still, the result is only slightly below the theoretically proposed second order convergence, and therefore very gratifying, indeed.

Table 5.1: Relative L_2 errors for different resolutions compared to a 800x800 reference grid

grid size	η_{U,L_2}	$order_U$	η_{p,L_2}	$order_p$
25×25	9.73e-02	-	1.42e+00	-
50×50	3.36e-02	1.53	5.72e-01	1.31
100×100	1.09e-02	1.62	2.26e-01	1.34
200×200	2.98e-03	1.88	8.30e-02	1.45
300×300	1.30e-03	2.03	4.20e-02	1.68
400×400	6.67e-04	2.33	2.36e-02	2.00

Strong scaling

Since the parallelization of CFD codes is indispensable in order to tackle interesting problems, we want to take a look at the performance of our code on a big CPU cluster, next. We want to see what kind of efficiency we can expect with our code in this regard. The subsequent study was performed on a parallel CPU cluster of the University of Bonn. In total, the deployed cluster *Atacama* consists of 1248 Intel Xeon CPU E5-2560 2.60 GHz cores, supplied by 78 PowerEdge M620 compute nodes. Each of these nodes comes with 16 CPU cores. The system is equipped with 4992 GB of memory, in

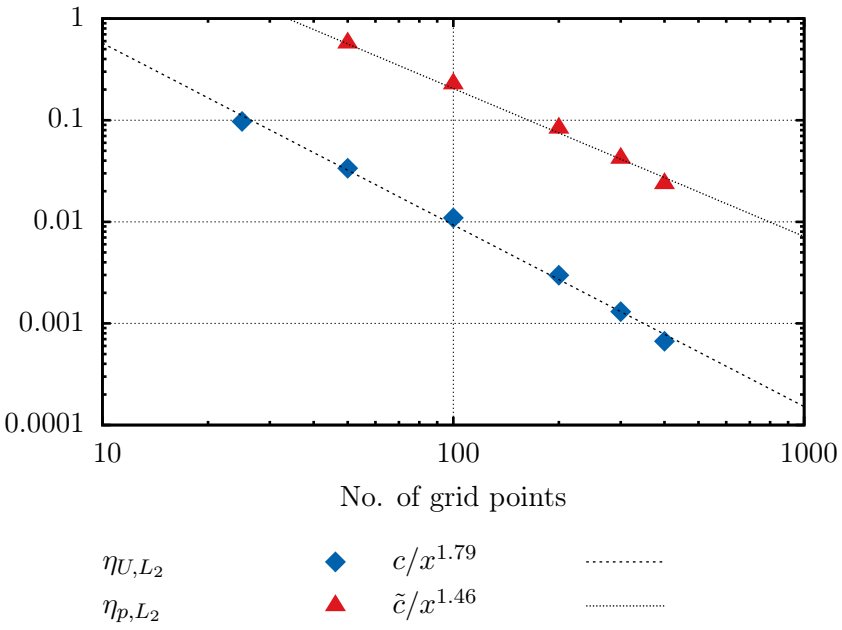

Figure 5.2: Progression of L_2 errors for the velocity U and the pressure p compared to the respective convergence order on a logarithmic scale

total, and the MPI communication routines are conducted by 56 GB/sec Infiniband connections. The cluster is operated by the Institute for Numerical Simulation and the Sonderforschungsbereich 1060 at University of Bonn and displays a Linpack performance of 20630 GFlop/s at a parallel efficiency of 80%.

For the following comparison, running times of the complete program execution were measured. To avoid static costs distorting the picture for higher numbers of processors, data outputs were disabled and the problem was chosen to be reasonably solvable for the total range of a single core up to the maximum of 512 cores we used. The conducted calculation is a now fully three dimensional driven cavity on a cubic grid with 256 cells in each direction, which amounts to ca. 16.8 Mio. cells in total. The simulation was run until a fixed time of $0.5s$ was reached. A clip through the corresponding velocity field is shown in Figure 5.1. The results of our study, concerning speedup, as well as the corresponding efficiency, can be seen in Table 5.2 and Figure 5.4. At a first glance, the observed efficiency is distinctly above 50 % up to a number of 256 cores and is thus already very reasonable. Looking more closely, especially the scaling above a number of 16 cores seems to be very good. Indeed, we can attest an efficiency of 80.2 %, when increasing

from 16 to 256 cores, a 16-fold increase, in relation to the 67.2 % we inherit from the comparison to a sequential run. We have to take into account here, that a full node of the employed cluster consists of exactly those 16 processor cores. In the end, only a comparison of a different number of such full nodes is a fair matter. Additionally, the interest in such a study is focused on the scaling for a large number of processors, anyway. On this front we can report very pleasing results. For the final run with 512 cores, the scaling seems to be a bit weaker than the very strong speedup before, but then also the running time of our experiment at this stage is already below three minutes, so we might be in a region, were the posed experiment is too small to make any definite statements. In conclusion, we can report a very sound scaling behaviour up to a number of 512 cores, with an efficiency of more than 80%, when increasing from 16 to 256 cores and a total speedup factor of 241.6. A further speedup, on even higher numbers of cores, with only slightly reduced efficiency, can hence be expected with very good confidence.

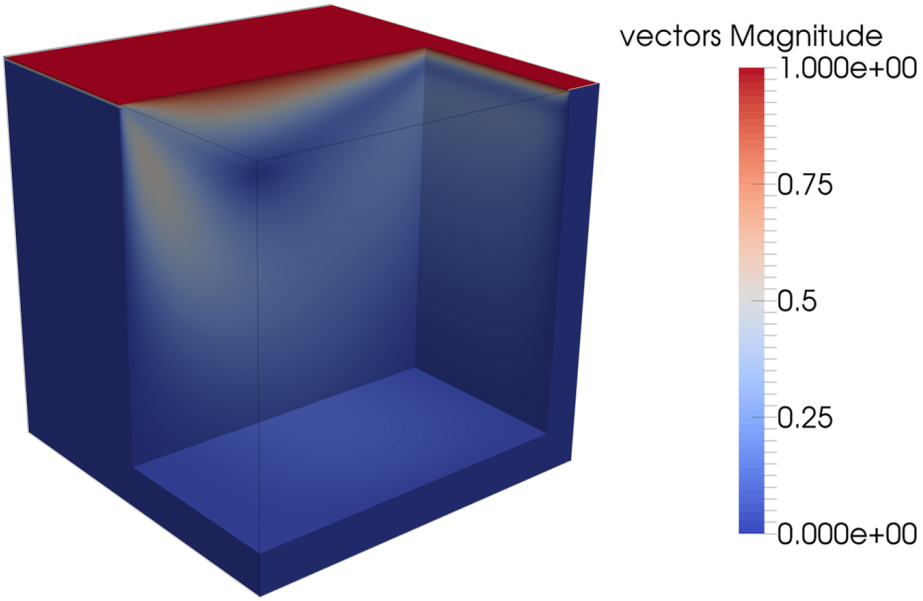

Figure 5.3: Clip through the velocity field of the conducted driven cavity experiment for strong scaling on a cluster.

5.2 A benchmark problem

We have already stated, that proving convergence of a CFD code of this complexity is currently out of reach. This becomes all the more true, when

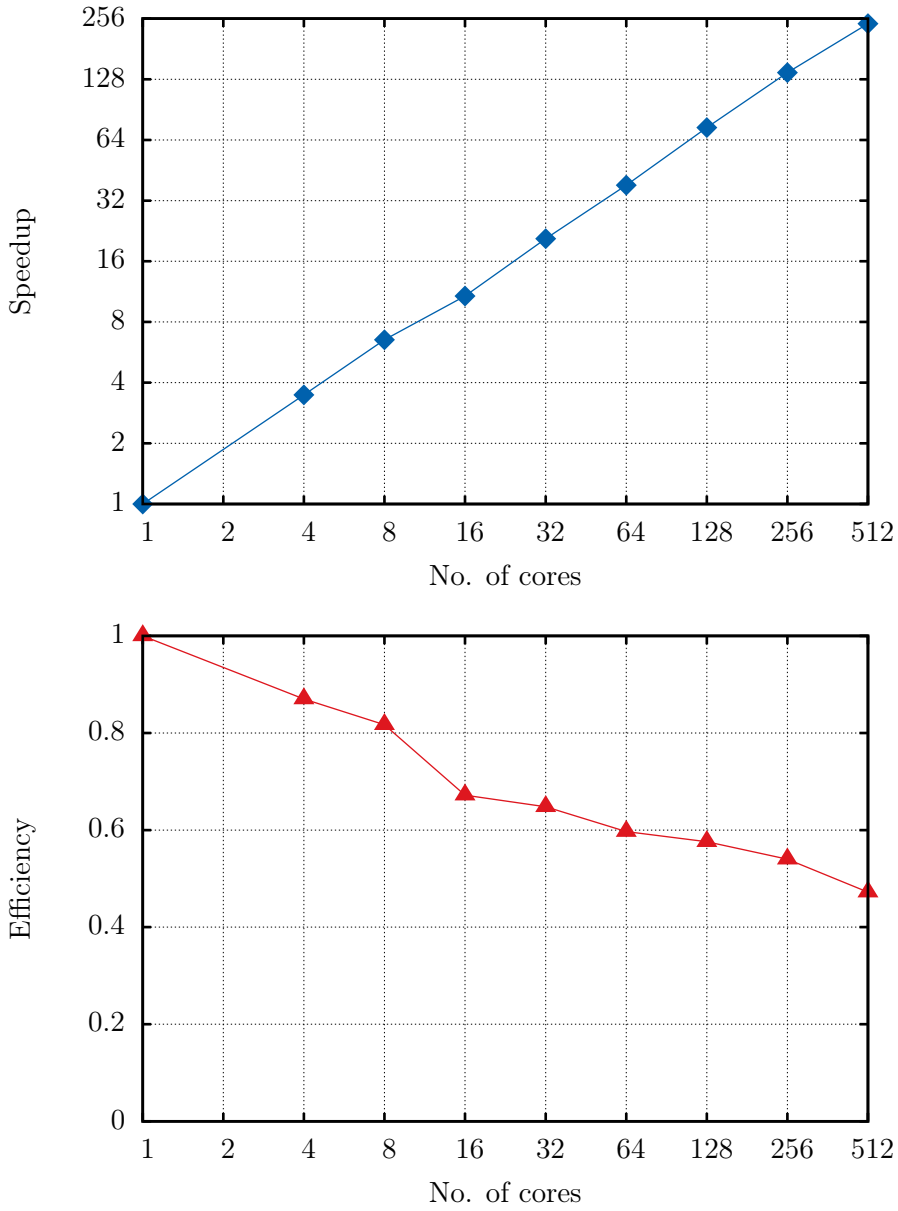

Figure 5.4: Speedup (top) and efficiency (bottom) plot for up to 512 cores on a CPU-cluster

Table 5.2: Speedup and efficiency of a three dimensional driven cavity simulation on up to 512 cores

no. of cores	absolute time [s]	speedup	efficiency
1	40981	1.0	100.0 %
4	11778	3.5	87.0 %
8	6273	6.5	81.7 %
16	3811	10.8	67.2 %
32	1977	20.7	64.8 %
64	1073	38.2	59.7 %
128	556	73.8	57.6 %
256	297	138.2	54.0 %
512	170	241.6	47.2 %

we consider more complicated experiments, like the external flow around an obstacle. Comparing the results of our algorithm to exact solutions is difficult, as well. Although some analytic solutions can be found in the literature (see i.e. [Whi91]), they are mostly restricted to the one dimensional case or don't solve the full Navier-Stokes equations, but some simplified version. Another approach for the verification of CFD codes is the Method of manufactured solutions (MMS) [OT02] which aims at manufacturing experiments specifically, such that a given solution emerges. The *AIAA Code Verification Project* [Ghi+10] lists examples of this method, but none of them seems applicable to our special situation. Yet, there are some tightly defined benchmark cases around, which allow the direct comparison to solutions of other CFD codes or physical experiments, that are conducted in wind or water channels. We want to see, if we can find something that suits our situation. Most compressible forms, like [LL87], focus on shock waves or other high Mach number phenomena, which are not in our interest or reach. What stays as an option, is to see if we can replicate results of incompressible benchmarks, if we resort to a small enough Mach number, by adjusting the speed of sound accordingly. This is what we want to try in the upcoming section.

Cylinder benchmark by Turek and Schäfer

We try to adopt a benchmark proposed by Turek and Schäfer for incompressible CFD codes. The test was first established in 1996 in [TS96], but the most recent results were published in [BMT12] in 2012. The proposed

task is to examine the flow around a cylinder of diameter D, positioned in a prescribed channel, at a Reynolds number of $Re = UD/\nu = 20$, where U is a characteristic velocity of the flow. As a means of evaluation, drag and lift forces, exerted on the body, ought to be calculated. A display of the exact setting, including the dimensions of the channel, as well as the position of the cylinder is given in Figure 5.5, which is adopted from Turek and Schäfer.

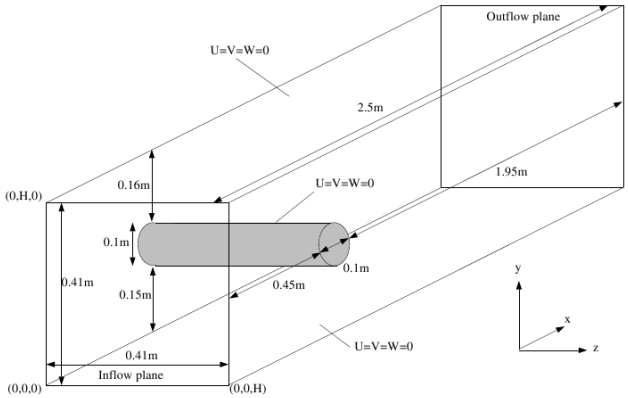

Figure 5.5: Setup and main dimensions of the 3D-1Z cylinder benchmark with circular cross-section according to Turek and Schäfer; taken from [TS96]

In the benchmark, a parabolic inflow profile according to the following equation is specified on the inflow plane:

$$\begin{aligned} u(0, y, z) &= 16 U_m yz(H - y)(H - z)/H^4, \\ v(0, y, z) &= w(0, y, z) = 0, \end{aligned} \tag{5.5}$$

where $H = 0.41\,m$ and $U_m = 0.45\,m/s$ denotes the maximal inflow velocity at the center of the channel. The characteristic velocity U is then

$$U = \frac{4}{9} u(0, H/2, H/2) = 0.2\,m/s, \tag{5.6}$$

which gives us the desired Reynolds number of 20. On the back of the channel a free in-/outflow condition is used, and on all other walls standard no-slip conditions are prescribed. Drag and lift forces, denoted by F_D and F_L, accordingly, are defined as follows

$$F_D = \int_S (\mu \frac{\partial v_t}{\partial n} n_y - p n_x)\, \mathrm{d}S \tag{5.7}$$

$$F_L = -\int_S (\mu \frac{\partial v_t}{\partial n} n_x + p n_y)\, \mathrm{d}S, \tag{5.8}$$

where n is the outer normal of the cylinder with components n_x, n_y and n_z. Since this is a special case of the components of equation (4.11), except for the extra term concerning the bulk viscosity, we use the same routines as for the bidirectional coupling. The corresponding dimensionless coefficients c_D and c_L, for completion, are then

$$c_D = \frac{2F_D}{\rho U^2 DH} \tag{5.9}$$

$$c_L = \frac{2F_L}{\rho U^2 DH}. \tag{5.10}$$

The integration in the expressions for the lift and drag forces are performed as described in Section 4.4. In addition we calculated the volume of the actual cylinder in our scene, to compare it to the magnitude of our deviations in lift and drag coefficients.

We performed calculations at different resolutions, as well as different values for the speed of sound. The exact setup is listed in Table 5.3. Since both resolution, as well as a higher speed of sound considerably increase computational cost, we cannot have both at an astronomically high level at the same time, but rather have to take compromises. To see what influence each of the parameters have, we tried both variants and want to compare the results. For all runs, we resorted to standard no-slip conditions on the cylinder, as the boundary condition presented in section 2.4 led to problems with stability.

Table 5.3: Grid size and speed of sound of the different simulation runs; × marks performed runs

grid size \ v_{Sound}	10	40	80	160
$250 \times 41 \times 41$	×	×	×	×
$500 \times 82 \times 82$		×		
$750 \times 123 \times 123$		×		
$1.000 \times 164 \times 164$		×		

Having declared the setup, we can now go forth and perform the calculations. First, we want to look at how good we can even approximate the cylinder with our current method. Therefore, we calculate the volume enclosed by the obstacle cells, that make up our cylinder. A comparison of these volumes with the correct volume of the cylinder is shown in Table 5.4.

Table 5.4: Rel. error of the cylinder volume at different grid resolutions

no. points in x	volume	rel. error
250	0.002184	32.2 %
500	0.002760	14.3 %
750	0.002931	9.0 %
1.000	0.003007	6.6 %
exact	0.003220	0

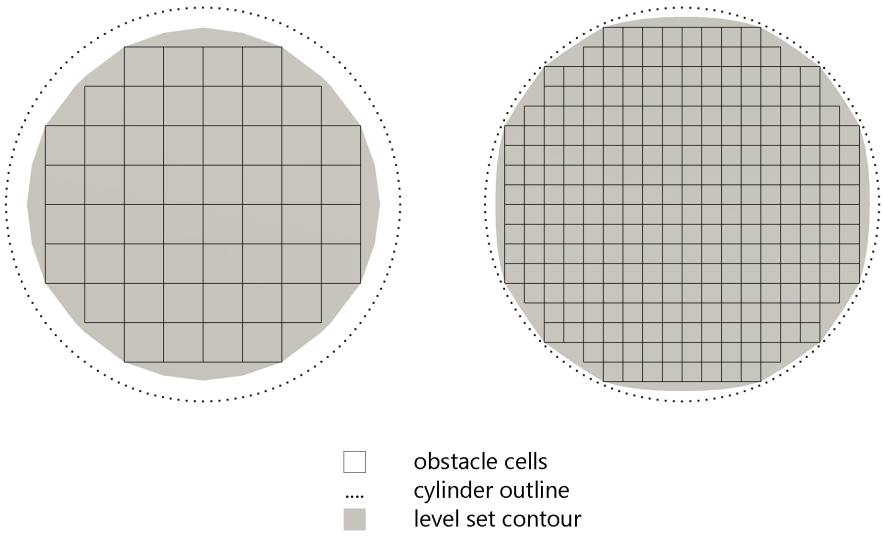

Figure 5.6: Approximation of the cylinder outline by the flag field and the level set contour at a resolution of 250x41x41 (left) and 500x82x82 (right)

We can see, that the error is quite big at the beginning at over 30 %. Even for a very fine resolution of 1.000 grid cells in the flow direction, which amounts to nearly 27 Mio. cells in total, we have an error in the representation of the volume, which is not negligible at 6.6 %. We want to see, if we can improve this behaviour. The way we create our level set function is to take all grid points, that are inside the object and create a characteristic function, which is then reinitialized. The problem is, that we underestimate the volume of our object, if the object is small compared to the resolution of the underlying grid. Figure 5.6 shows this very plainly, by comparing the resulting object at a resolution of 250 and 500 grid cells in the flow direction. For calculating drag and lift forces this is not desirable, since they will be equally small. What we could do instead, is to take the points closest to the cylinder as its surface, but tracking these points in 3D is quite tedious, especially for more complex objects. We will try a more simple approach, which gives quite similar results, but consists of simply enlarging the geometry by $\frac{\sqrt{2}}{2}$ cells in each direction, or, in turn subtracting $\frac{\sqrt{2}}{2}dx$ from the level set function and updating the flag field accordingly. Due to the dependence on the cells width, the additional term goes to zero when refining the mesh, so we still have convergence. As portrayed in Table 5.5 though, the error in the volume is much lower to begin with (10.4 %), and even the progression seems to be better when increasing the resolution. It looks like we now have second order convergence in the volume. The difference between the two methods is depicted in Figure 5.7 for a resolution of 250x41x41, where the improvement is quite obvious. For our finest mesh, we are left with an error of 0.8 % which is quite reasonable, considering that we use an underlying uniform mesh.

Table 5.5: Rel. error of the cylinder volume computed with the refined method, motivated by closest points

no. points in x	volume	rel. error
250	0.002886	10.4 %
500	0.00312	3.1 %
750	0.003173	1.5 %
1.000	0.003194	0.8 %
exact	0.003220	0

We now want to turn our attention to the results of our simulations. The flow was simulated for 16 s in all runs, after which no changes were observed

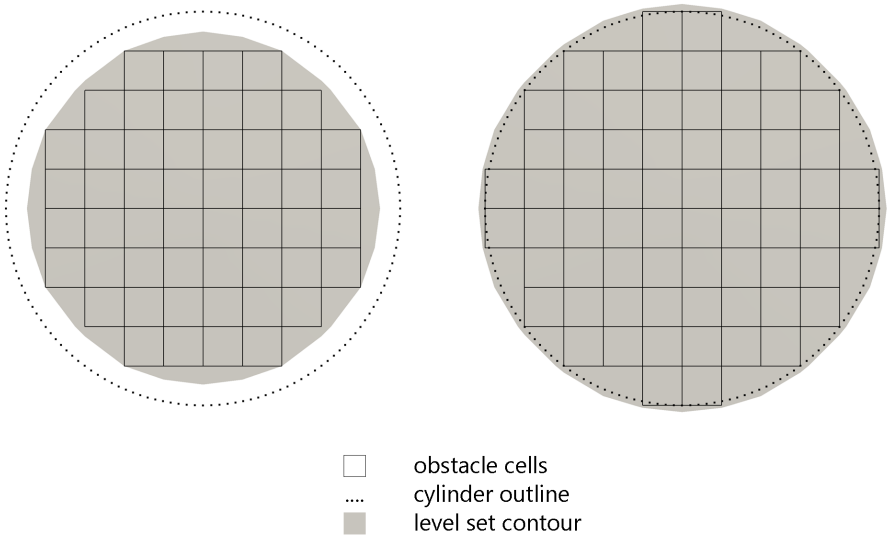

Figure 5.7: Comparison of the cylinder approximation by all inner points (left) and approximation to closest points (right) at a resolution of 250x41x41. The relative error in the volume drops from 32.2 % to 10.4 %.

and steady-state could be safely declared. The emerging velocity field for $v_{sound} = 50\,m/s$ on a 500x82x82 mesh is depicted in Figure 5.8 and is visually indistinguishable from the results of Bayraktar et al. Following the discussion above, we want to examine the influence of the speed of sound and also the resolution of our grid on the sought figures. We split these experiments in two series, each of which investigates the influence of one of the parameters. Each series consists of 4 runs, with one of them being shared among both series as a starting point.

Let us first consider the speed of sound. The results for drag and lift coefficients, as defined in Equations (5.7) - (5.10), are listed in Table 5.6. Allthough run (a) already starts with a Mach number M below 0.1, which is typically considered to be in the incompressible domain, we can see, that there is still an influence of v_{sound} on the result. In general, there seems to be a trend for both c_D and c_L among the runs.
With increasing speed of sound, except for the starting value of 6.186, drag coefficients rise steadily, going up to a value of 6.218. While the last value is slightly overshooting the data given in the literature, we shouldn't overrate this effect, since relative errors for all of the values at a maximum of 0.5 % are already far below the rel. error of 3.1 % for the flag field volume, which is very good. Coming to the lift coefficient, the picture presents itself differently.

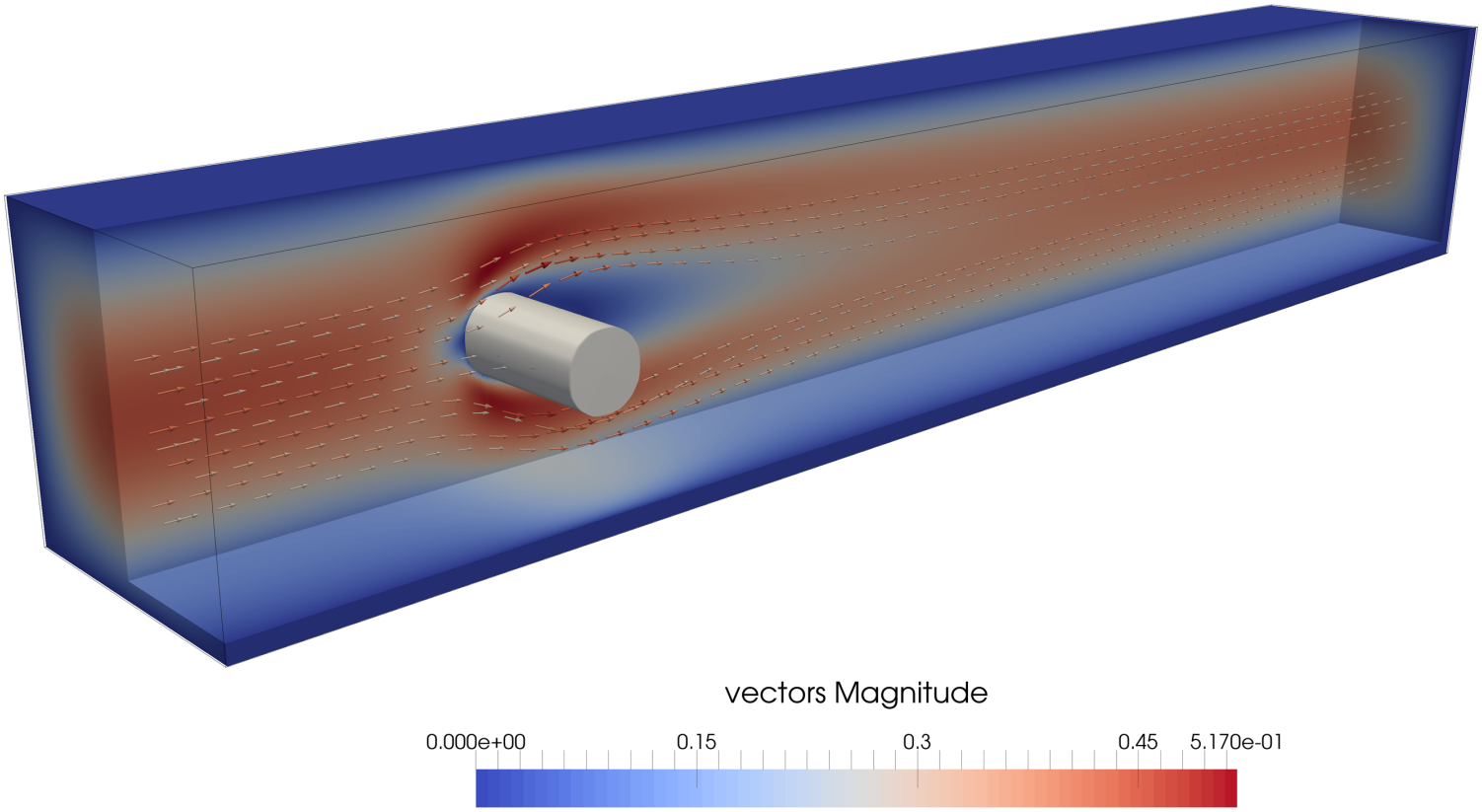

Figure 5.8: Velocity field of the flow around the cylinder after 16 s with a parabolic inflow profile according to Equation 5.5

Again, there is a trend, this time, though, seemingly going in the wrong direction. Even worse, while the absolute error is still small (around 0.015), the relative error is quite huge, and we cannot even correctly predict the sign of the coefficient, indicating downforce instead of lift. However, anticipating the results of the second series of experiments, we tend to conclude that the given resolution of 500 points in the main direction of flow, is just not up to the task for such a sensitive variable, depending on very small influences, indeed. Even though the number 500 seems rather large, at first glance, we have to remember, that the cross section of the actual cylinder is only covered by a square of 20x20 cells as depicted in Figure 5.6. More important is the general observation, that in both cases, the values seem to stabilize, with only small changes between the last two runs. Even the run at 40 m/s seems not far off the final result. Therefore, we took this run to see in our second series if we can improve it's coefficients by altering the resolution. Finally, we should keep in mind, that changing the speed of sound, not only, changes the compressibility of our fluid, but also demands smaller time steps in our calculations. These time steps, in turn, might introduce dispersive errors at the typical speeds appearing in our case. While a Mach number of around 0.1 should be totally applicable, the validity of using much lower values, like in our study, has to be questioned and was mainly done in this experiment, to spot its effect. At least in this scenario, no obvious harming effect can be assessed, since the values seem to converge with increasing speed of sound.

Table 5.6: Comparison of drag and lift coefficients at different values for the speed of sound and 250 grid points in the flow direction

run	grid size	v_{sound}	c_D	c_L
0	Turek & Schäfer	-	6.185	0.00941
a	$500 \times 82 \times 82$	10	6.186	0.00049
b	$500 \times 82 \times 82$	40	6.177	-0.00537
c	$500 \times 82 \times 82$	80	6.186	-0.00735
d	$500 \times 82 \times 82$	160	6.218	-0.00773

For the resolution, things are kind of reversed, and the results put the values of our first series somewhat into perspective. Looking at the drag, we see that the values decrease with rising resolution. The first run indicates, that a lower resolution, has a rather large negative effect, but we still stay inside the 10.4 % reference, we gave ourselves. Going, further up, however, has only small consequences, obviously tied to the already very good values. The runs (3) and (4) show a slightly lower value than given by Turek and

Schäfer, with the last one indicating, that it will probably stay in this region, even for higher resolutions. In the end, our error is somewhere close to 1 %, which is totally fine. Remembering, that we expect the values to still slightly increase for higher speeds of sound and the two effects might potentially cancel, we are very pleased with the outcome. Going over to the lift, we can see that the values for higher resolutions are much improved over the ones seen before. Now, we can correctly predict a lifting force on the cylinder, albeit a bit smaller than we hoped for. Also, while the drag coefficient seems to stabilize among the runs and only varies slightly, the lift coefficient still shows quite high deviations even for the higher resolutions. Considering, however, the small total error, which is now in the order of 0.005 and the fact, that this is also the instance, where deviations among the compared codes in the literature is highest, we can still be very satisfied with our results.

Table 5.7: Comparison of drag and lift coefficients for different grid resolutions at $v_{Sound} = 40\,m/s$

run	grid size	v_{sound}	c_D	c_L
0	Turek & Schäfer	-	6.185	0.00941
1	$250 \times 41 \times 41$	40	6.609	-0.03401
2	$500 \times 82 \times 82$	40	6.177	-0.005374
3	$750 \times 123 \times 123$	40	6.103	0.001716
4	$1.000 \times 164 \times 164$	40	6.114	0.004191

In total, we can say that our results are pretty satisfying. Even though our code is at first glance not perfectly suited for the presented drag and lift calculations, as they heavily depend on correctly resolving the flow at or near the boundary, which profits vigorously from having some form of adaptivity, we can still assert a very solid performance in computing both lift and drag values, if we can cope with the mandated high resolutions. We can not totally replicate the results of Bayraktar et al. but we are in as close proximity, as can be expected.

5.3 A physical problem

After successfully solving some benchmark problems, which are mainly constructed for CFD code verification purposes, we now want to look into solving an actual physical problem with realistic constants and parameters, that also involves more complex geometry. Since we cannot expect to accurately resolve turbulence at high Reynolds numbers, we looked for an example,

where the Reynolds number is kept below a threshold value of approximately 10.000. One area, where these requirements are met, is the flight of insects. Due to their small bodies, Reynolds numbers tend to be in a range of 3.000-5.000. Small insects use active flying techniques with high frequencies of wing beat, though, but butterflies, in contrast, also show some periods of gliding. Therefore we decided, to set up a model of a typical butterfly and see what we can get out of it. The model is considered to be a proof of concept, only, and was therefore kept relatively simple. To be able to resolve the wings accurately at a reasonable resolution, the thickness of the wings was emphasized to 0.3 mm. The model at hand has a wingspan of approximately 38 mm. The four symmetric wings are attached to a body of 18 mm length. A drawing of the exact model, as well as an isometric view can be seen in Figure 5.10 and Figure 5.9. The model is positioned in the anterior half of a fluid channel of length 0.1 m and is inclined at an angle of attack of 12°. It is simulated on a grid consisting of 192 Million cells, which is by far the biggest and computationally most expensive case in our work, and clearly shows the backdrop of a uniform mesh without the possibilities of adaptivity. The exact setup is listed in Table 5.8. The inflow is set to $0.8\,m/s$, which we imagine to be a typical velocity encountered by such an insect, even though their maximal air speeds can reach much higher values.

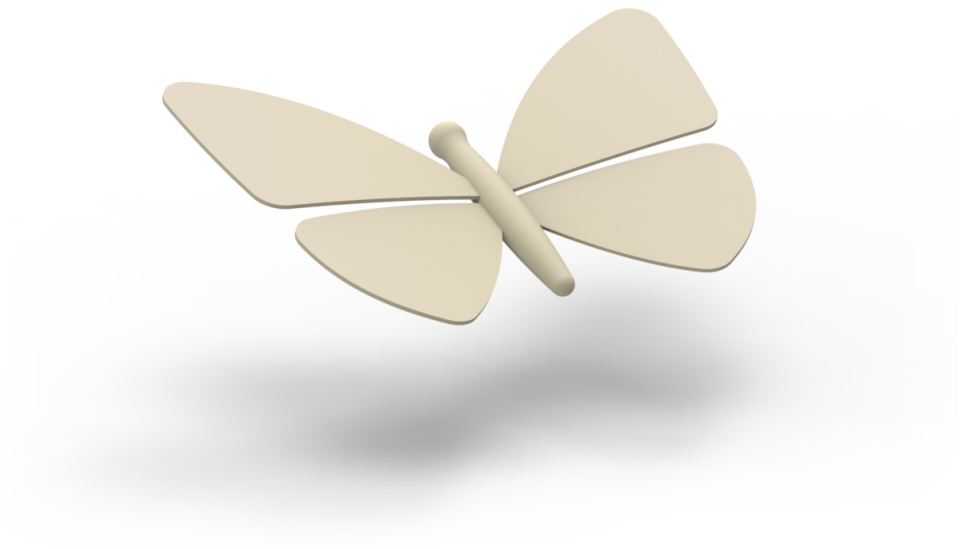

Figure 5.9: Isometric view of the simulated butterfly model; Rendered in KeyShot 5 Student Edition

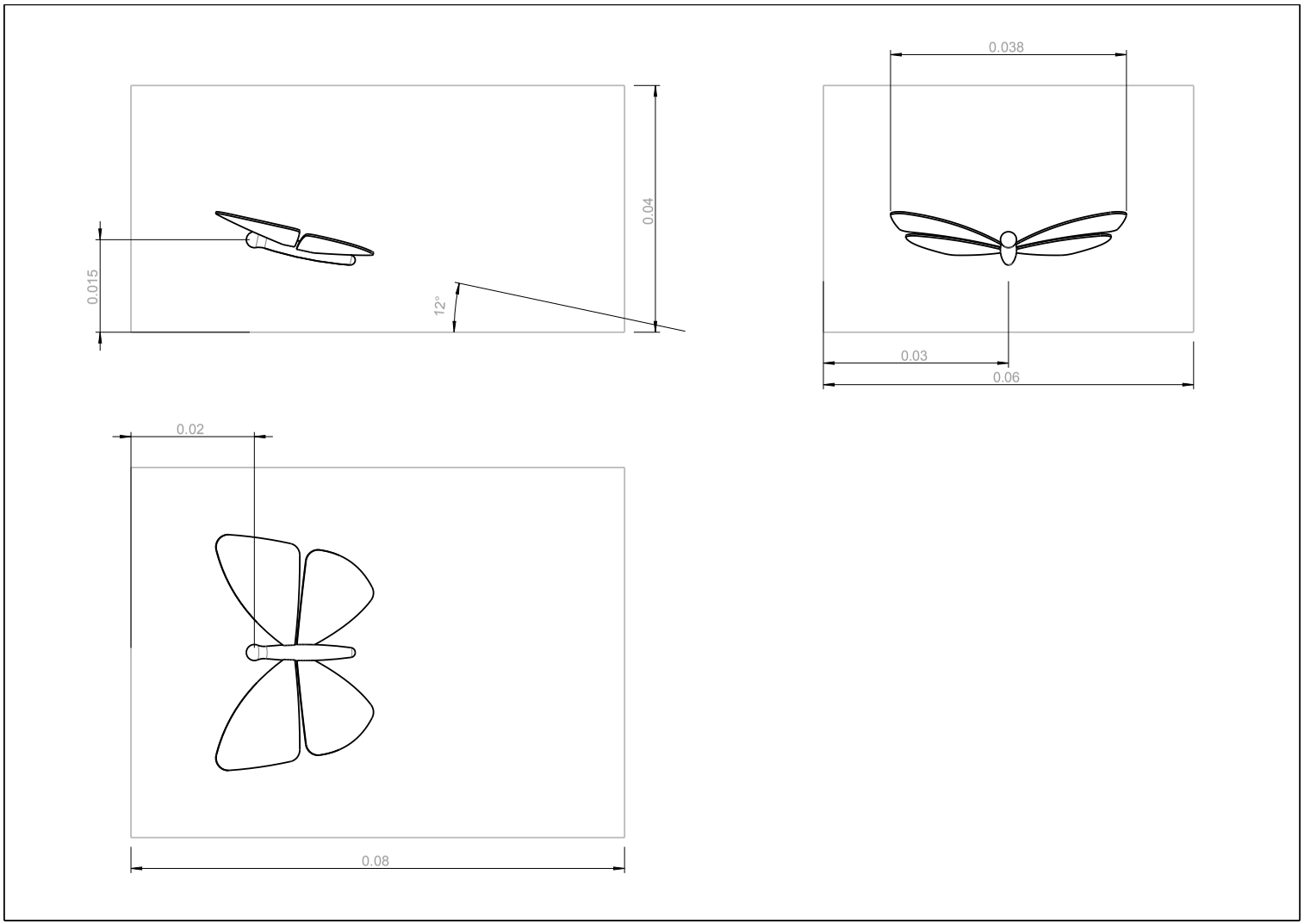

Figure 5.10: Drawing of the considered butterfly model and its placement in the flow channel in top front and side view; Modelled in Pro/ENGINEER Wildfire 5.0 Student Edition

Table 5.8: Overview of the most important simulation parameters for the conducted simulation of a butterfly

parameter	x	y	z
length $[m]$	0.08	0.04	0.06
no. of points	800	400	600
inflow $[m/s]$	0.8	-	-
v_{Sound} $[m/s]$		10.0	
ρ $[kg/m^3]$		1.2	
μ $[Pa{\cdot}s]$		1.8e-06	
Δt $[s]$		3.4e-06	

For our fluid, we try to simulate air, as closely as possible, hence, we set a viscosity of 1.8e-06 Pa·s and a static density of $1.2\,\mathrm{kg/m^3}$. Our only concession concerns the speed of sound, where we chose a value of 10 m/s instead of the theoretical 340 m/s, yet, at the comparably small fluid velocities appearing, this gives us still a Mach number of ca. 0.1. Going even lower than this should make only a very small difference in terms of compressibility, but costs us heavily in terms of computational efficiency. Due to the quite staggering amount of points involved, we already face a tough restriction concerning the allowed time steps. In practice, we ended up with a value of approximately $\Delta t = 3.4\text{e-}06\mathrm{s}$, which means in turn, that something like the equivalent of 300.000 time steps have to be executed in order to reach a simulated time of just one second. Although the simulation doesn't reach steady-state, at least in the given time-frame, due to the enormous amount of data arising, we restrict ourselves to the analysis of a single time step in this case, which is at 0.2 s into the simulation. Also, we had to revert to standard no-slip conditions, here, since we couldn't find a stable setup with the more advanced boundary conditions presented in Section 2.4.

Due to the conducted simplifications, the results of our simulation are meant to be understood as a proof of concept, rather than a detailed study. Therefore we don't want to over-analyse the resulting flow field, but we still take a brief look at the most basic features. In Figures 5.11 and 5.12 we present both velocity and pressure distributions around three different slices of the butterfly. We can see, that there is a nice region of high pressure, at least under the front half of the butterfly. On top of it, a zone of lower pressure reaching over the whole length of the insect can be found, which is even extending to the last part of the lower side. This gives us a first

indication, that the (simplified) wings actually produce some lift. In the velocity plots, we can also see, that there is a huge separation going on in the middle half of the butterfly, that probably stems from the narrow connection to the clumsy body towards the middle. In contrast, the flow around the outside of the wings is much cleaner, but the leading edge radius of the wing seems to be too small, for having an attached flow round the upper side of it. This is no surprise, however, since air is rather viscous at the size of such animals, and therefore, the main means of movement is by wing beat, anyhow. Their flight circumstances are probably more comparable to swimming, rather than the flight of a large bird. Figure 5.13 shows another view from the side and top of the butterfly. Employing some stream tracers, we can see the typical tip vortices forming at the end of the wings, compensating for the pressure difference between upper and lower sides of them. Their inward rotation, gives us yet another hint at the lift generated by our butterfly. As a final notice, we are pleased to see, that although at a restricted flight regime, we are able to actually make relevant experiments, in physically simulated air.

5.4 Unidirectional fluid-motion coupling

There are many interesting problems and real-world applications, that involve some kind of moving objects, that can only be solved using a method like the one at hand. In this section, we want to give examples for some unidirectional coupling of fluid flow with body movement.

Oscillating ball

First, we want to start off with an example of unidirectional coupling. We consider a ball oscillating vertically in a horizontal flow field. If we denote the velocity of the centre of mass of said ball with u_M, v_M and w_M accordingly, the movement of the ball is determined by the equations

$$\begin{aligned} v_M(t) &= 0.5\cos(2t) \\ u_M(t) &= w_M(t) = 0, \end{aligned} \tag{5.11}$$

representing a sinusoidal oscillation with a total movement range of 1 m and a period of π seconds. The ball itself was chosen to have a diameter of 0.4 m and its centre of mass (CoM) is located at coordinates (2.0,1.0,1.0), to begin with. For the horizontal flow, we used the parabolic inflow described in Equation (5.5) again, with a maximal velocity of $U_m = 2.0\,m/s$ this time around. The setup of the surrounding channel, as well as the most important simulation parameters, is shown in Table 5.9. Given this information, we can

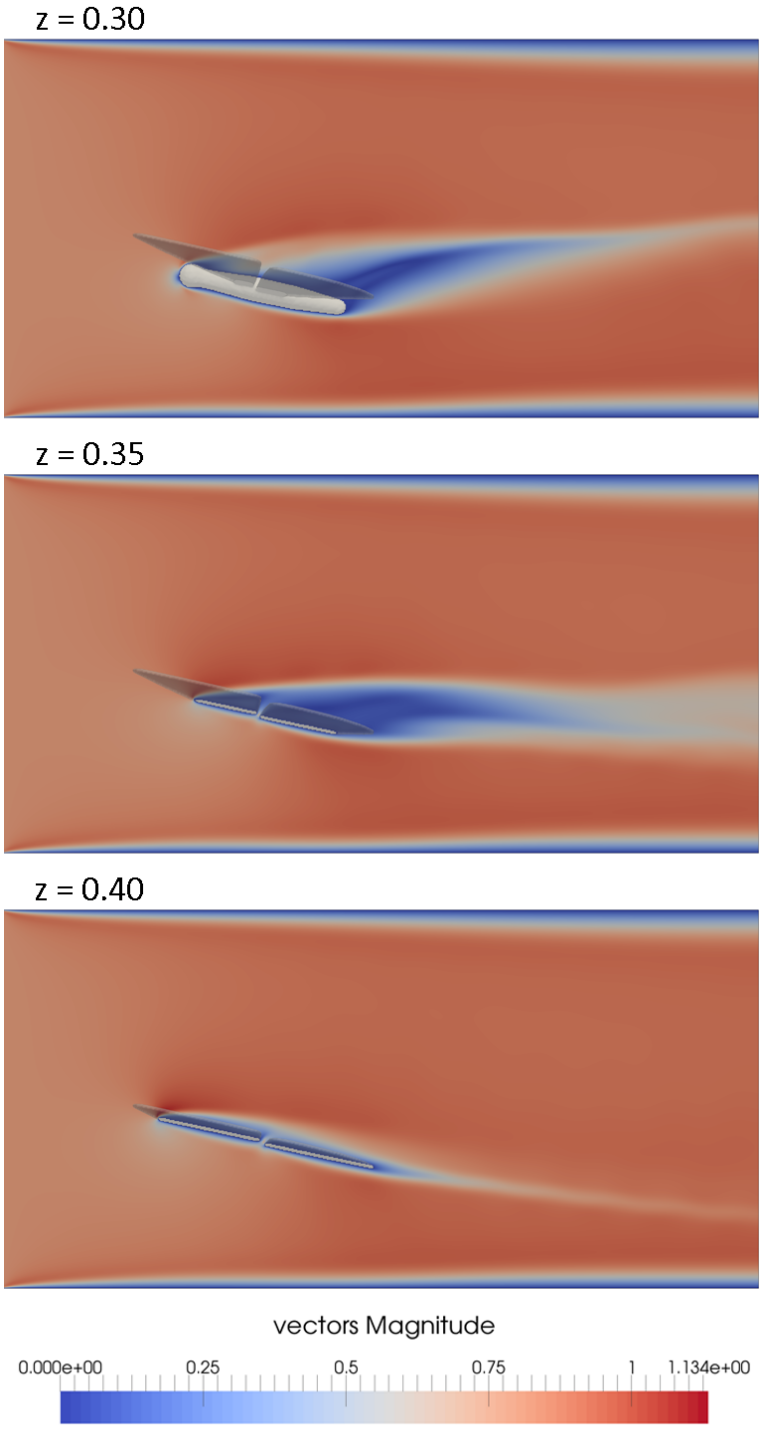

Figure 5.11: Slices of the velocity field at $z = 0.03\,m$, $z = 0.035\,m$ and $z = 0.04\,m$ with remaining butterfly (opaque)

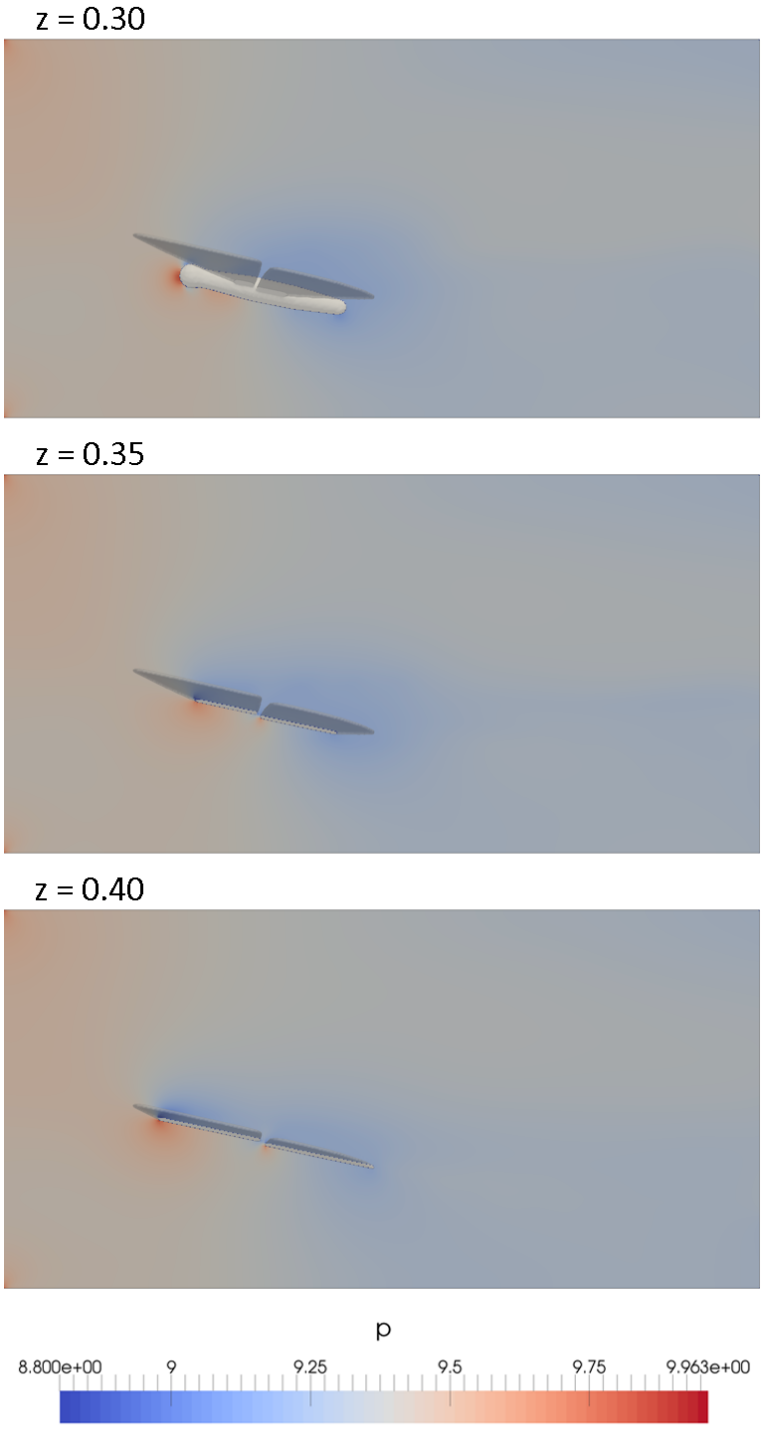

Figure 5.12: Slices of the pressure field at $z = 0.03\,m$, $z = 0.035\,m$ and $z = 0.04\,m$ with remaining butterfly (opaque)

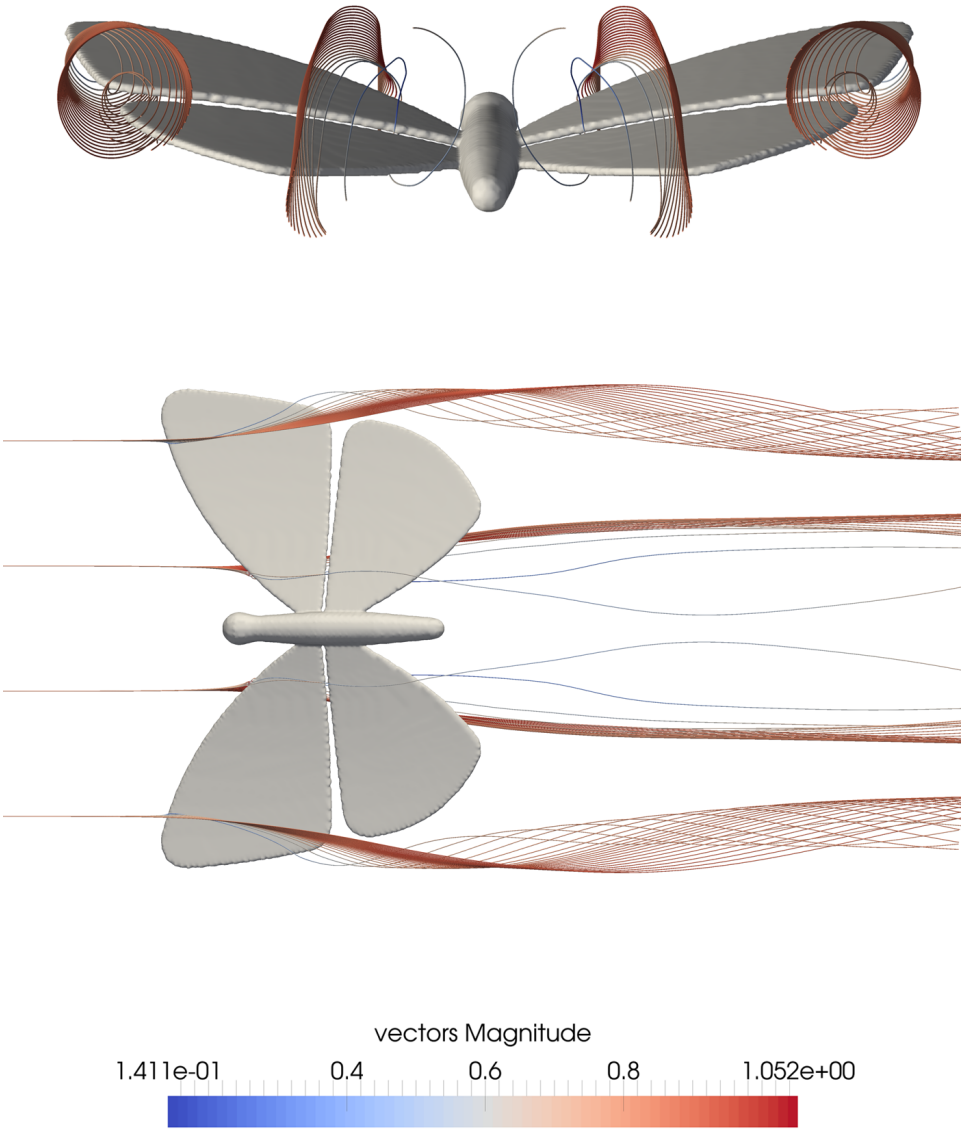

Figure 5.13: Tip vortices at the wings as seen from the back and bottom of the butterfly; visualized with stream tracers emerging from vertical lines

calculate the Reynolds number as

$$Re = \frac{\rho \cdot U \cdot l}{\mu} \approx 70, \tag{5.12}$$

where we took the diameter of the ball as our typical length l and the mean inflow velocity $U = \frac{4}{9}U_m \approx 0.9m/s$ as the reference in terms of velocity.

Table 5.9: Overview of the most important dimensions and simulation parameters for the oscillating ball

parameter	x	y	z
length $[m]$	8.0	2.0	2.0
no. of points	360	90	90
inflow U_m $[m/s]$	2.0	-	-
CoM $[m]$ $(t = 0)$	2.0	1.0	1.0
v_{Sound} $[m/s]$		20.0	
ρ $[kg/m^3]$		1.0	
μ $[Pa{\cdot}s]$		0.005	
Δt $[s]$		$\approx$ 3.5e-04	

With the layout of the simulation in place, we can now look at the resulting flow field. Figure 5.14 shows a clip through the catholicity field at $z = 1.0\,m$ after the simulation reached a periodic or quasiperiodic state. At the back of the ball a long tail has emerged testifying the sinusoidal movement. The three different pictures in the figure represent the transition from upward movement to downward movement at the upper turning point. In turn the tail changes from a lower arc behind the ball to an upper arc. At this particular Reynolds number no vortex street can be found in the trail and the overall impression is very tidy, just the way we expected it. At higher Reynolds numbers the picture appears more cluttered and the specific features quickly become much harder to track. We want to take a look at this behaviour in a different setting, next.

Rotating ball

In this simulation, we want to consider a rotating ball in a cavity. Although our method doesn't allow for rotating objects per se, as mentioned in Section 4.3, it is obviously possible to do it in the case of a rotationally symmetric object by moving it in a circular motion around the rotation centre. There is

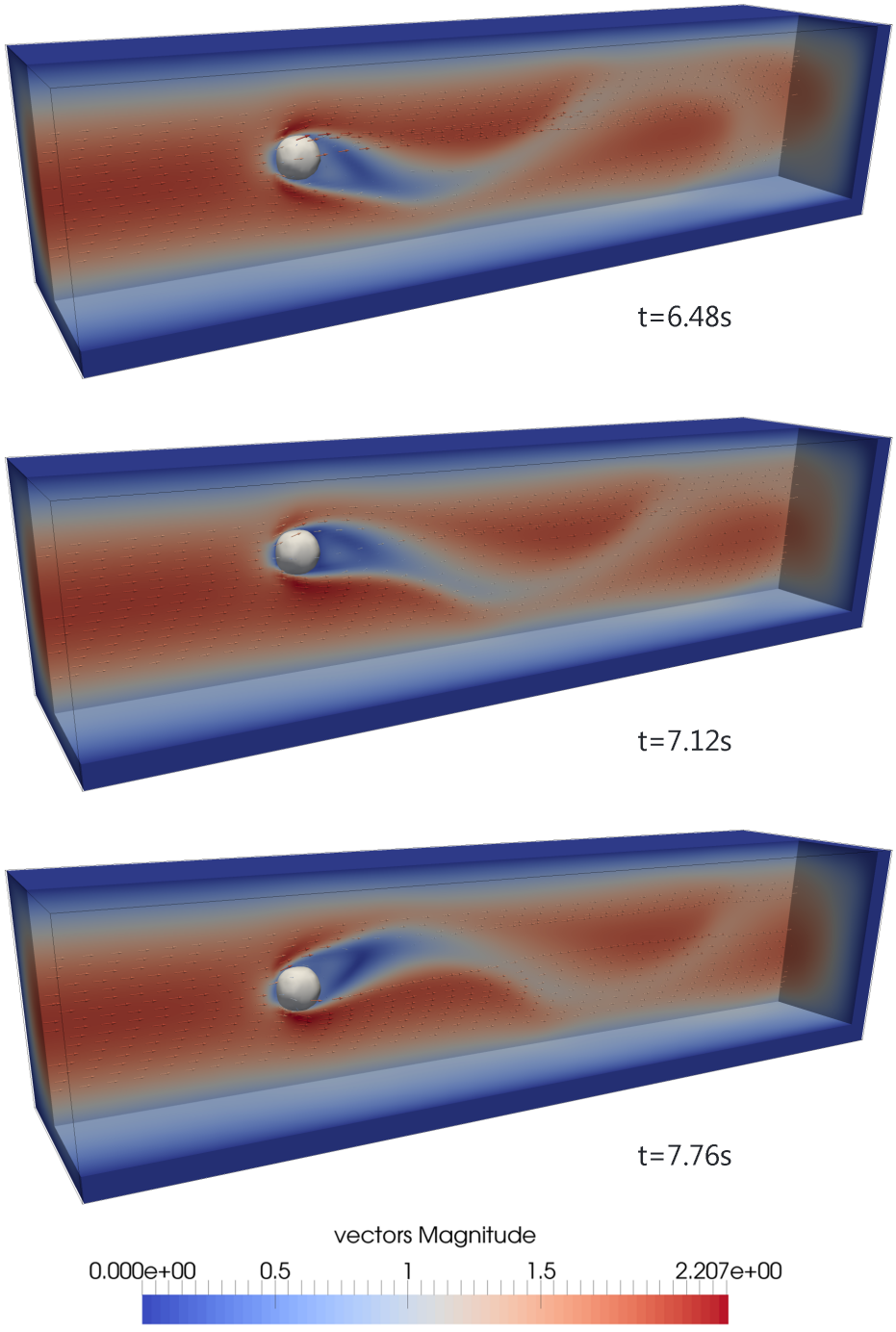

Figure 5.14: Velocity field of the oscillating ball in horizontal flow at three different time steps; before, in close proximity and after the upper turning point of the motion

no inflow in this case, all the fluid movement is generated by the movement of the ball. The movement of the ball, which is placed in the same starting position, is described by the following set of equations:

$$\begin{aligned} u_M(t) &= 2\cos(2t) \\ v_M(t) &= 2\sin(2t) \\ w_M(t) &= 0 \end{aligned} \tag{5.13}$$

Together they make for a circular motion of diameter $D = 2.0\,m$ and a period of π seconds. We want to perform the simulation at two different viscosities this time around, namely at $\mu = 0.002\,Pa{\cdot}s$ and $\mu = 0.0005\,Pa{\cdot}s$, respectively. If we use the magnitude of the movement $|\boldsymbol{u}_M| = 2.0\,m/s$ as a reference velocity and again, the diameter of the ball, i.e. 0.4 m, as reference length, we can calculate the Reynolds number Re as 400 and 1600, correspondingly. In Table 5.10, once again, the most important simulation parameters are outlined.

Table 5.10: Overview of the most important simulation parameters for the rotating ball

parameter	x	y	z
length $[m]$	4.0	4.0	2.0
no. of points	180	180	90
inflow $[m/s]$	-	-	-
CoM $[m]$ $(t = 0)$	2.0	1.0	1.0
v_{Sound} $[m/s]$		10.0	
ρ $[kg/m^3]$		1.0	
μ $[Pa{\cdot}s]$		0.002 / 0.0005	
Δt $[s]$		$\approx$ 6.0e-04	

Turning our attention to the results, which are depicted in Figures 5.15 and 5.16, we make the following observations. Compared to the oscillating ball above, instead of the dead water behind the ball, the trail area now makes up for most of the activity. This is obviously due to the movement of the ball now being the only source of fluid flow, compared to the static background flow in the above experiment. Again, the path of the ball can be very nicely traced in the shape of its trailing velocity field. Already at a Reynolds number of 400, we can see vortices forming in this trail area and

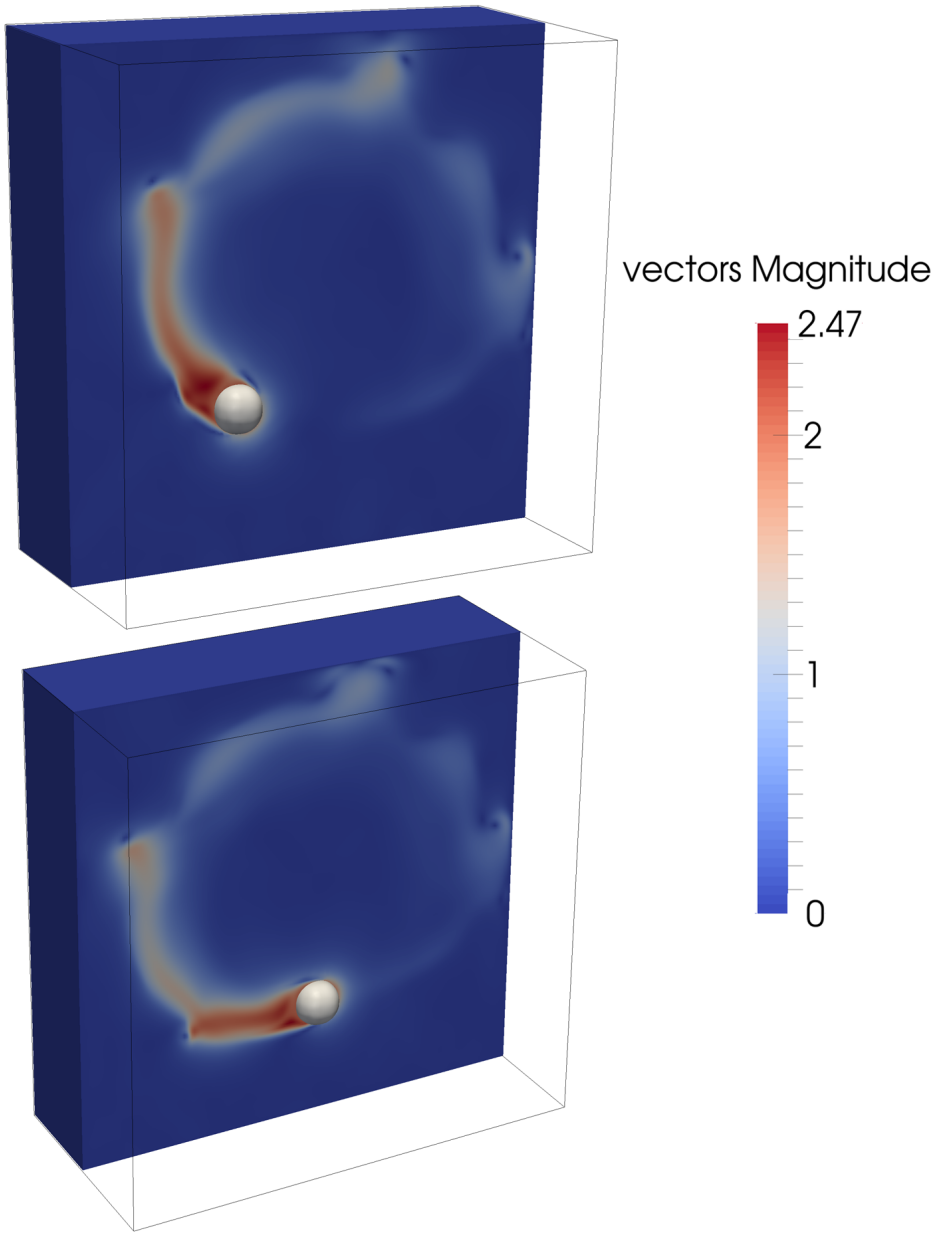

Figure 5.15: Velocity field and level set contour of the rotating ball at $t = 2.8\,s$ (top) and $t = 3.2\,s$ (bottom) and Reynolds number $Re = 2000$

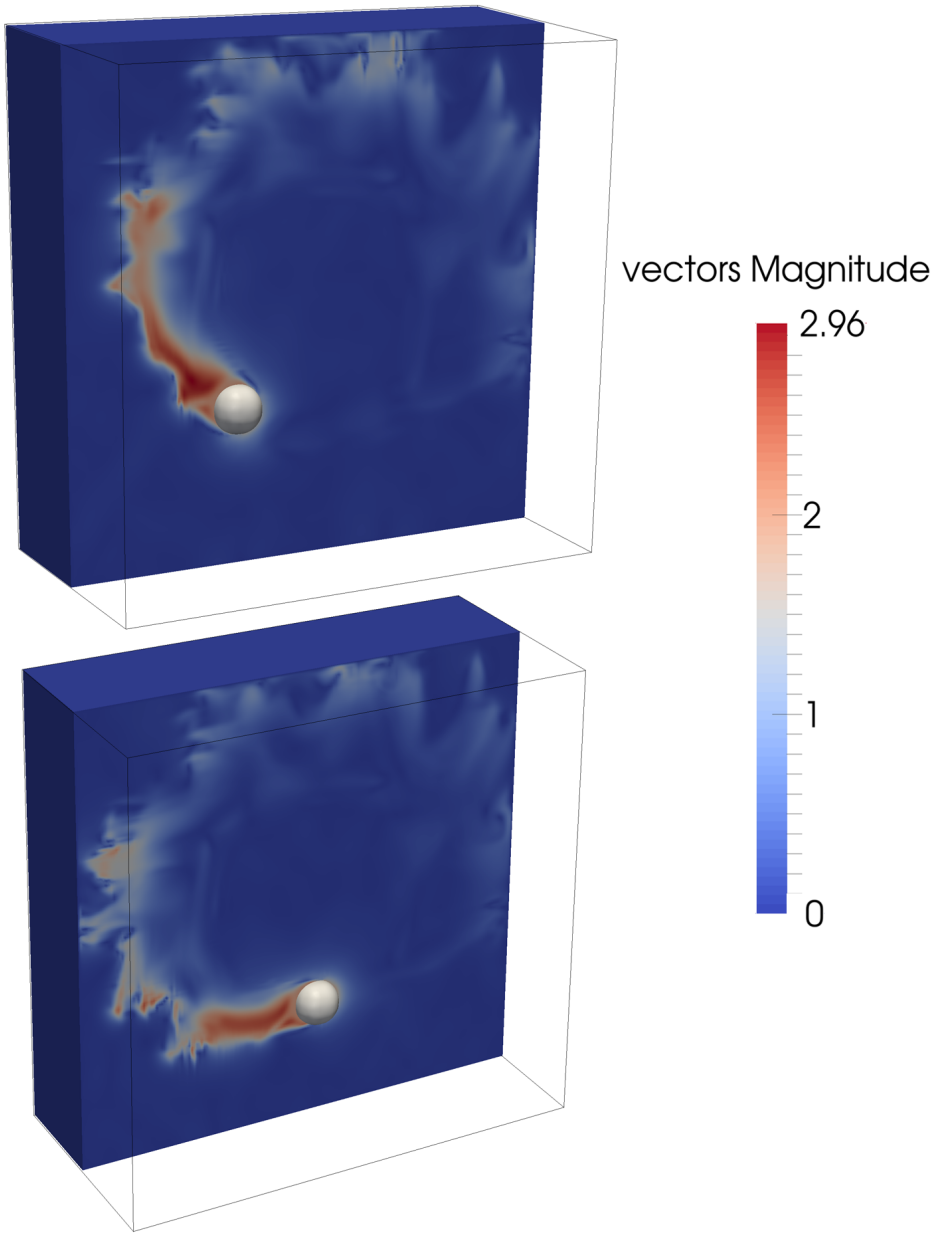

Figure 5.16: Velocity field and level set contour of the rotating ball at $t = 2.8\,s$ (top) and $t = 3.2\,s$ (bottom) and Reynolds number $Re = 8000$

moving to the border of the cavity in regular intervals. While at Re 400 we have very distinct vortices forming over a period of time, at Re 1600 the picture is a bit different. The flow still has the same basic structure, but now the vortices break apart into many small disturbances as soon as they form. Overall the picture is very cluttered. It's difficult to say if the small scale flow features can still be accurately resolved by our mesh, or if the turbulence is already high enough to require a much finer grid. Additionally, the maximal velocity in the trail rises from just below 2.5 m/s to nearly 3 m/s, although the general shape and length of the trail seem to be very similar. In principle, this is the expected behaviour, though, as a smaller viscosity should mean less fluid is being pulled behind the ball, but at a faster maximum velocity.

5.5 Bidirectional coupling

Having seen two different cases of unidirectional fluid-motion coupling, in this section, we want to try the possibilities of bidirectional coupling, i.e. of objects, that not only influence the surrounding fluid, but are now also directly driven by the flow. We consider these as examples of what possibilities there are, always keeping in mind the restrictions mentioned in Chapter 4.

Movement against the stream

With our first example of bidirectional coupling, we want to stay in the same setting as before, considering a ball in a channel. The basic setting is very similar to the oscillating ball discussed above, the only difference being, that we now only define an initial velocity of the ball and subsequently letting the forces, generated by the flow, act freely upon it. One can imagine the experiment as a ball being thrown against the wind, being slowed down by it, and eventually changing direction and coming back to its initial position. With this in mind, we want to take a look at our simulation parameters (compare Table 5.11), that were chosen in a way to accommodate this whole process in our given channel. We start of with our ball in the middle of the channel at coordinates (4.0,1.0,1.0) and a starting velocity of -2.0 m/s in x direction. The fluid is initially at rest and has a relatively high viscosity of 0.01 Pa·s to ensure that no detachments or other assymetries occur behind the ball, which would push it our of the centre of the channel after the turning point is reached. For this experiment, we also fall back to the standard uniform inflow condition at the left boundary of the channel. In addition to the usual parameters, for the bidirectional coupling, we have to prescribe a density for the solid object on which our fluid acts. In this case we set the density to 30 kg/m^3, which gives us a reasonable time and distance, until the eventual change of direction happens.

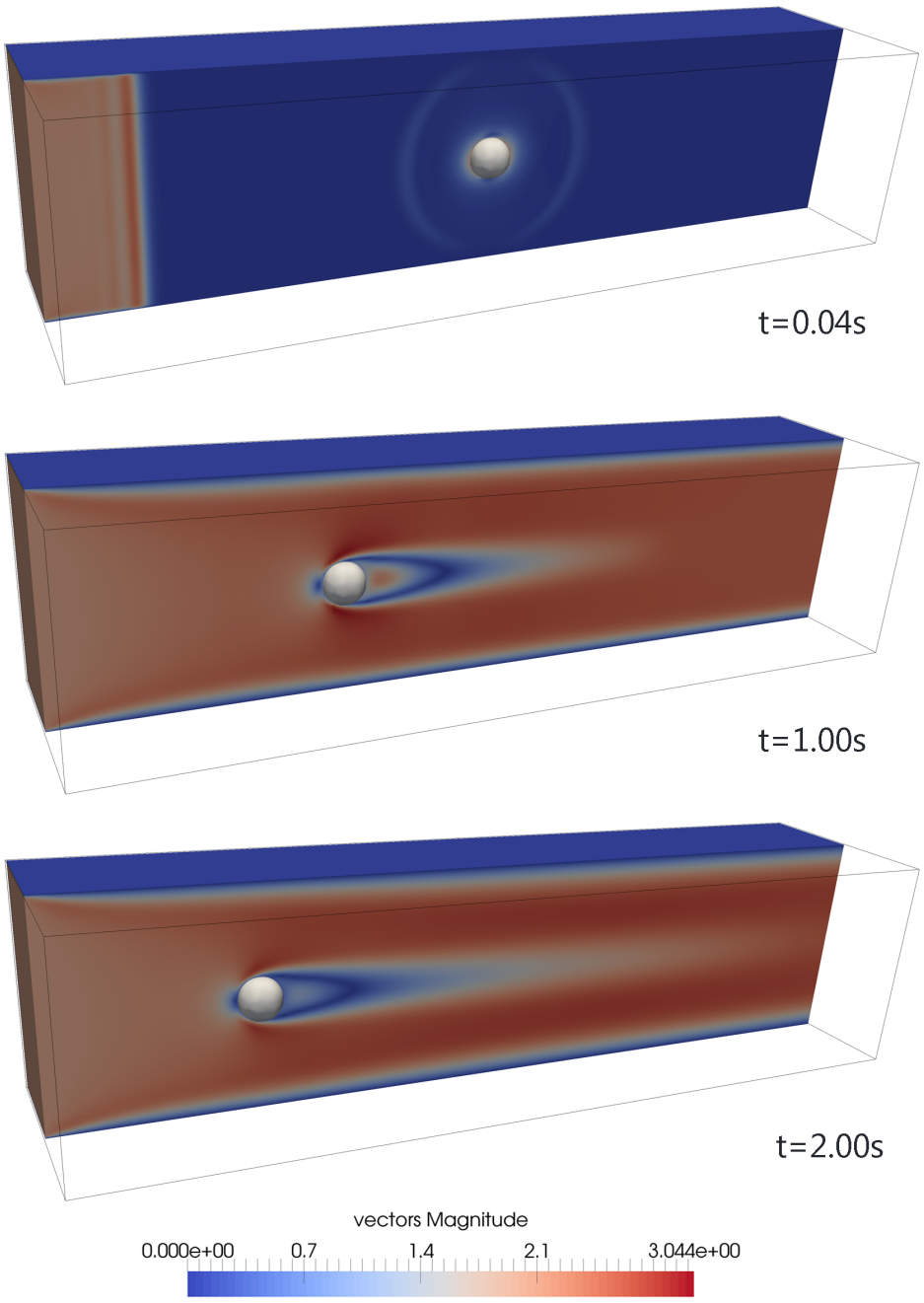

Figure 5.17: Velocity field and level set contour of a ball moving contrary to the flow at $t = 0.04\,s$, $t = 1.00\,s$ and $t = 2.00\,s$ and being slowed down by it until near rest

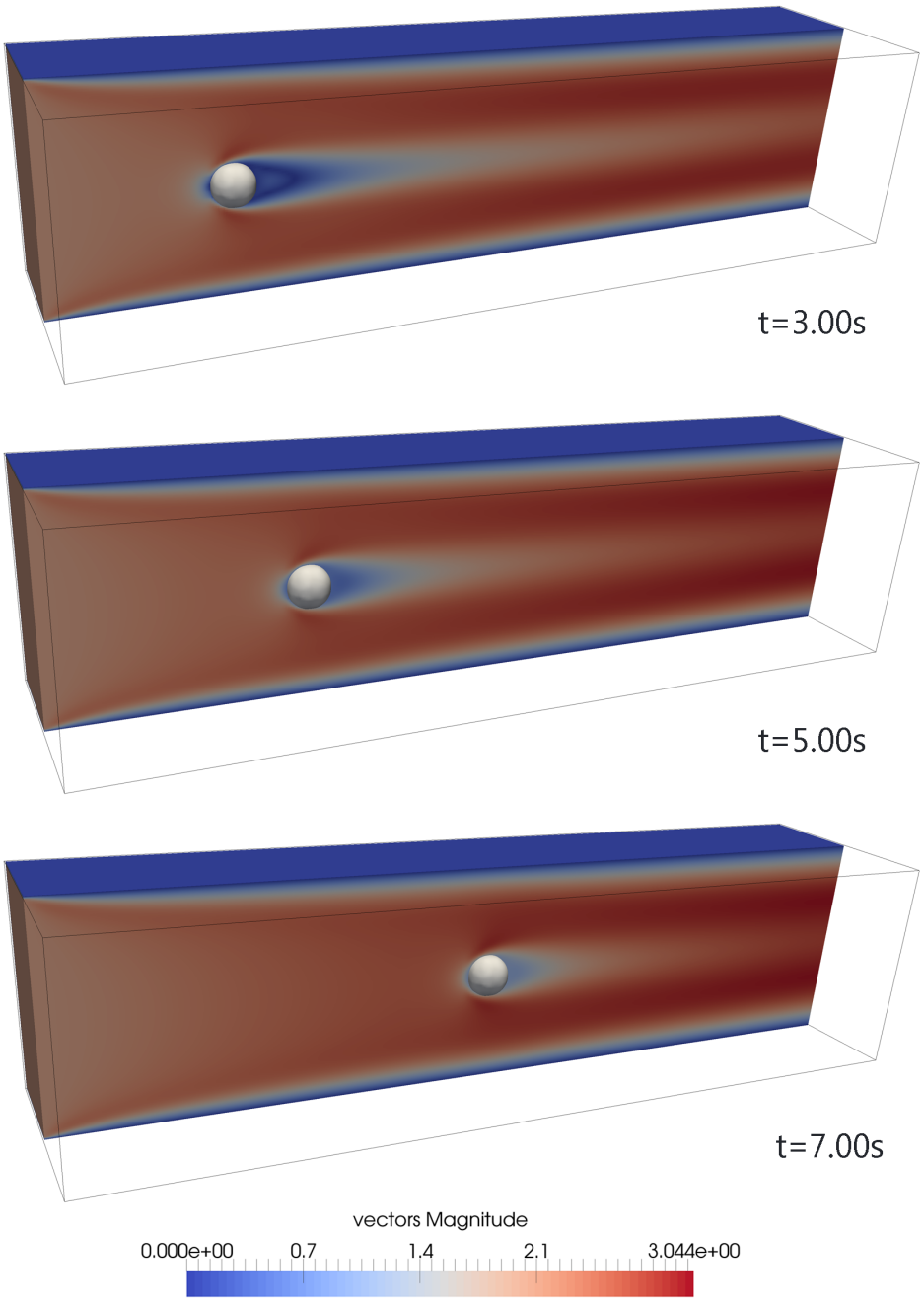

Figure 5.18: Velocity field and level set contour of the ball at $t = 3.00\,s$, $t = 5.00\,s$ and $t = 7.00\,s$ after reaching the turning point and being dragged back by the flow

Table 5.11: Overview of the most important simulation parameters for the movement against the stream case

parameter	x	y	z
length $[m]$	8.0	4.0	2.0
no. of points	360	90	90
inflow $[m/s]$	2.0	-	-
CoM $[m]$ $(t = 0)$	4.0	1.0	1.0
v_{Sound} $[m/s]$		20.0	
ρ $[kg/m^3]$		1.0	
ρ_s $[kg/m^3]$		30.0	
μ $[Pa{\cdot}s]$		0.01	
Δt $[s]$		$\approx$ 3.6e-04	

The chronological order of the events is depicted in Figures 5.17 and 5.18, showing once again the velocity field on a clip through the centre plane of the channel. The first picture shows the initial position of the ball right in the centre of the channel, just after the experiment started. Due to the sudden initial movement of the ball, as well as the inflow, we see some sound waves emerging, that are reflected at the boundaries, but dampen out over time. The second picture is already one second into the experiment. Throughout the length of the channel the fluid is already moving at close to the inflow velocity, with the usual increase at the level of the ball, due to it blocking part of the channel. Behind the ball (to the right in the pictures), the typical drop-shaped trail for low Reynolds numbers has emerged. Looking closer, however, we can spot another trail inside the bigger one, stemming from the motion of the ball itself, that pulls the fluid in its own direction of movement contrary to the surrounding flow. At two seconds, this inner trail already begins to vanish, indicating that the ball has slowed down quite significantly. Glancing at Figure 5.18 we can see that it has nearly reached the maximal displacement on its trajectory. In contrast, the outer drop is still extending its reach to the back of the channel. At three seconds, we have reached the turning point (the exact time is 3.05 s) at coordinates (1.54,1.00,1.00), corresponding to a distance of 2.46 m from its initial position. The two layered drops have merged into one, and the ball starts its voyage back to its origin. From here on the drop-shape begins to shrink, as the ball is accelerated in the opposing direction. At $t = 7.00\,s$ it has already gone past the middle of the channel and is headed towards the outflow with a velocity of 1.07 m/s, which is slowly crawling up to the 2.0 m/s of the surrounding flow. The deflection

in the cross-section of the channel is still below 1 mm in both the y and z direction. In total, we can attest a first successful test of our bidirectional coupling as described in Section 4.4. The observed results, all seem to be very reasonable, at the least.

Flying butterfly

As a final example, we want to put something together showcasing everything we have seen so far packed up in one simulation. We have already seen a very complex geometry, with our butterfly model. Once again, we want to emphasize on the quite ridiculous amount of over 190 Mio. cells being deployed, here. Being able to still run such an experiment in a reasonable time frame of around two to three days gives testimony to the quite insane computing powers enabled by modern hardware, as well as the efficiency of our code, in dealing with such a grid. Up to this point, we have only seen the butterfly stationary, now, we want to combine this, with the bidirectional coupling introduced above, and test, if it produces enough lift to keep itself up under gravity. In short: let us see if it can fly. We therefore introduce a constant gravity acceleration of $-9.81\,\mathrm{m/s^2}$ in y-direction, additional to the forces generated by the flow. For the density of the butterfly, we chose a value of $400\,\mathrm{kg/m^3}$, resulting in a mass of $\approx 0.1\,g$ for the whole model, which seems plausible for a butterfly of this size. The rest of the setup stays as described in Section 4.4 for the stationary case, except for the inflow being turned up to 2.2 m/s this time. Looking at the previous numbers, this should give us the kind of lifting forces needed to overcome the gravity, or at least for gliding only slowly to the ground. As a result, of course our Reynolds number goes up to a value of approximately 5000, which should result in a much livelier flow field than before. To stay in the same kind of region, concerning the Mach number we also increased our speed of sound to 20 m/s, dealing with the fact, that this further decreases our time steps to around 1.9e-06 s.

Now, when activating the forces making up our coupling right from the beginning, the butterfly starts to fall down right away, as there is no time for the flow to build up any significant forces counteracting it, and there is no way they will be able to catch it once it reaches a certain velocity. To adjust to these circumstances, we initially simulate statically for a period of 0.05 s, and only enable both the gravity, as well as the fluid forces after this initial, neutralized phase. Having declared the setup of our final experiment, we now want to answer our initial question: Can the wings actually produce enough lift to make the butterfly stay in the air?

Looking at the resulting time line depicted in Figures 5.19 and 5.20, we can answer the question. In the second picture, we can see that the butterfly

Table 5.12: Overview of the most important simulation parameters for the flying butterfly

parameter	x	y	z
length $[m]$	0.08	0.04	0.06
no. of points	800	400	600
inflow $[m/s]$	2.2	-	-
v_{Sound} $[m/s]$		20.0	
ρ $[kg/m^3]$		1.2	
ρ_s $[kg/m^3]$		400.0	
μ $[Pa{\cdot}s]$		1.8e-06	
Δt $[s]$		1.9e-06	

indeed moved slightly upwards (as well as to the back) from its initial position. Looking at the output, we see a lifting force of around 110 % the force enacted by gravity. Looking at the following time steps, though, we see, that after this initial rise, the model comes back down again, slowly sliding closer to the ground. This behaviour is quite natural in our setting. The longer our simulation goes on, the faster our butterfly becomes in the flow direction, therefore decreasing the difference to the velocity of the fluid myself. With this effective air speed of the butterfly shrinking, obviously, the lift produced by its wings also drops and our model starts to go back down. Looking a bit closer at the actual flow around the insect, we can also very clearly detect the predicted effects of the increase in the Reynolds number. There are a lot of vortices separating from the body part of the butterfly giving it a far wilder look compared to before. In total, we are very pleased with the outcome of this experiment, proving the very good stability, we actually have with our coupling.

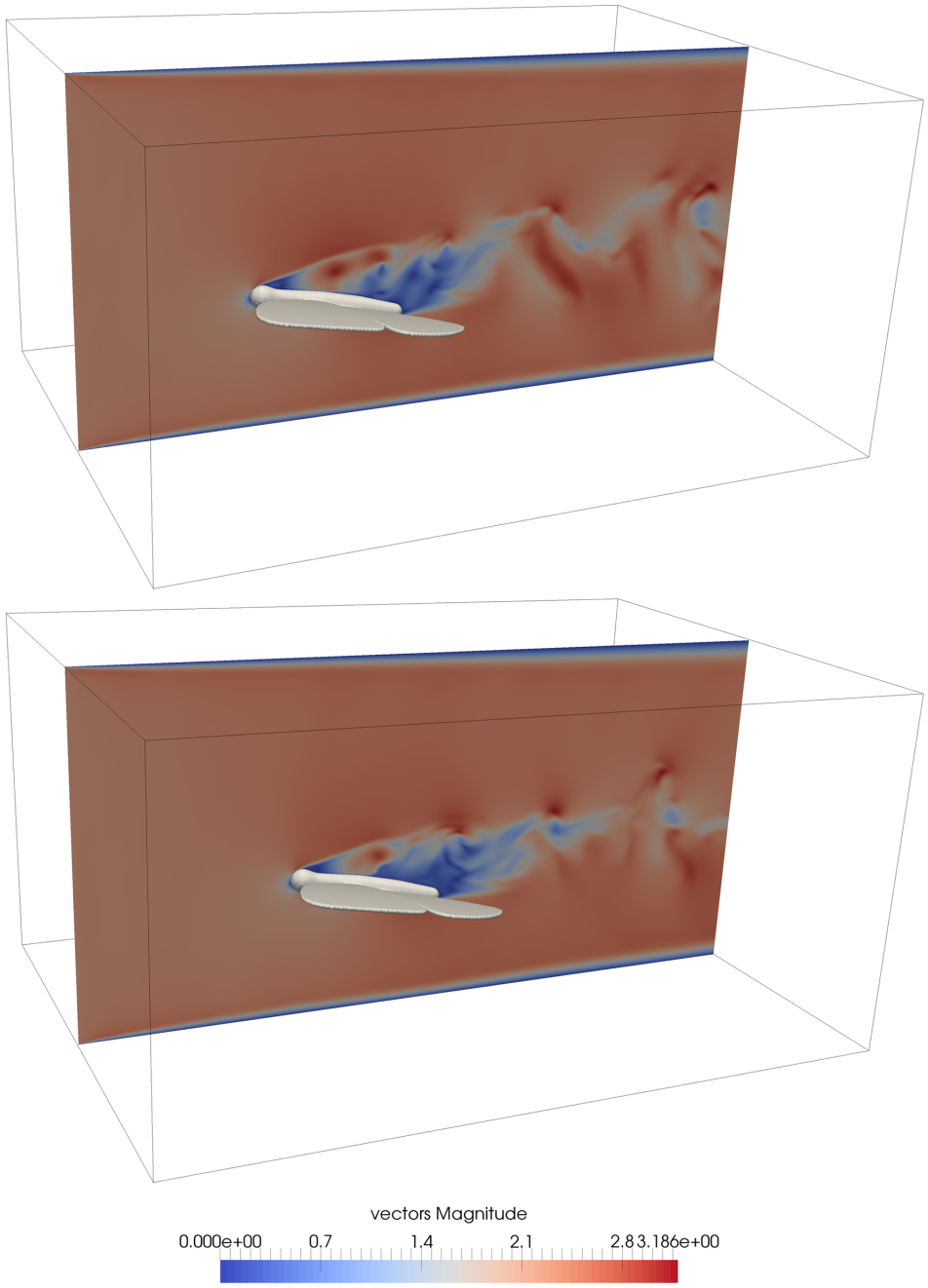

Figure 5.19: Velocity field and level set contour of the flying butterfly at $t = 0.05\,s$ (top) and $t = 0.10\,s$ (bottom)

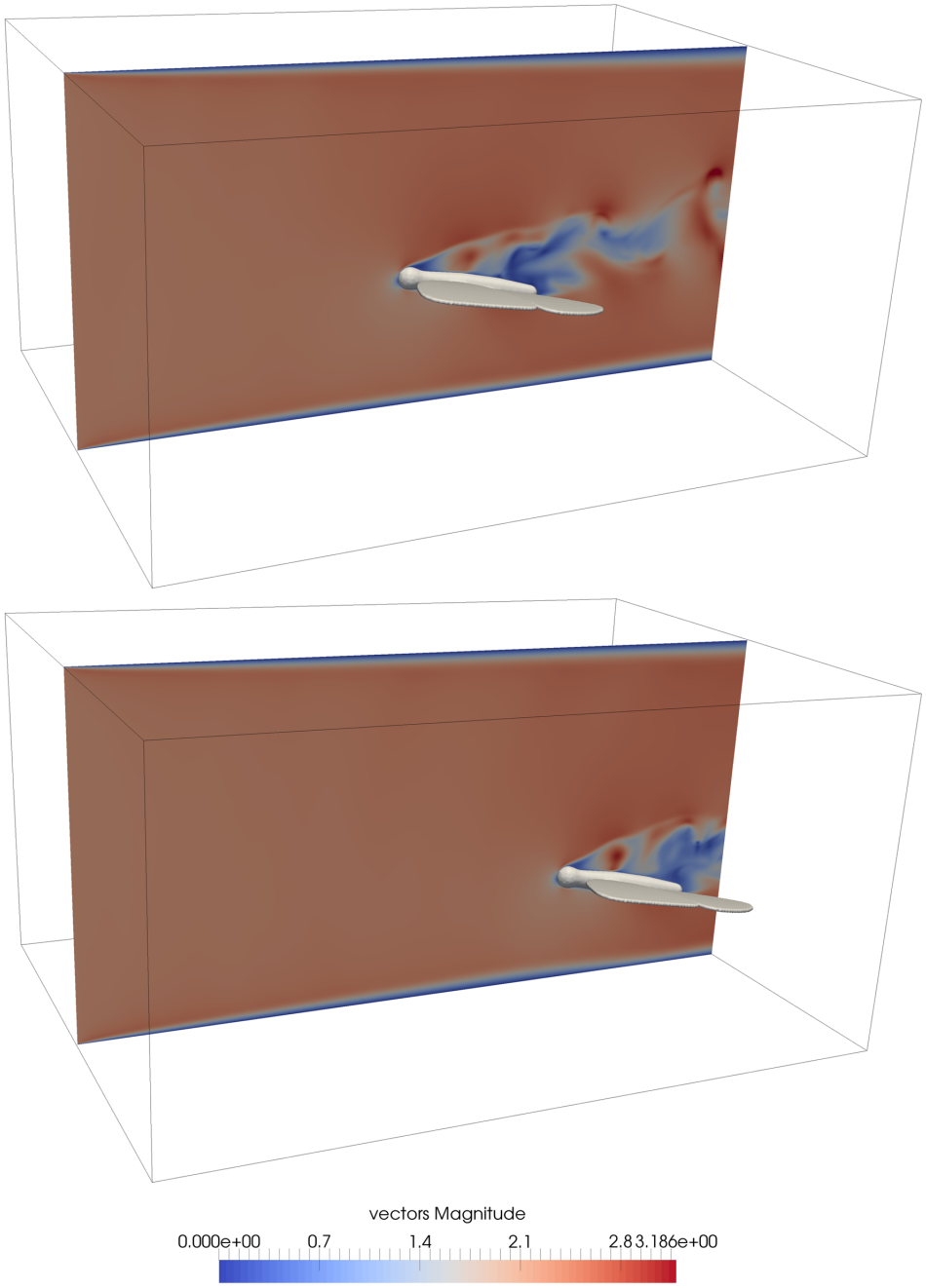

Figure 5.20: Velocity field and level set contour of the flying butterfly at $t = 0.15\,s$ (top) and $t = 0.20\,s$ (bottom)

Chapter 6

Outlook

Summary

The goal of this work has been to examine the behaviour of the unidirectional, as well as the bidirectional coupling of of a fluid simulation to moving obstacles, in a compressible setting. Indeed, we can report a very stable performance of our implemented code, especially concerning initial conditions. Even sudden movements, at the beginning of a simulation, like in our first example of the bidirectional coupling, only lead to some expected pressure waves emerging from the object, superimposing our picture, until they eventually dampen out. In conclusion we have implemented an efficient, explicit, compressible flow solver, that meets the demanded requirements. Complex geometry can be directly imported via stl models from external software, and is translated to our level set approach. In our computational benchmarks, when approximating a fine reference solution, we could show an order of convergence of 1.79 and 1.46 for the velocity and pressure fields, respectively. With a parallel efficiency of 80.4 % at a 16-fold increase of the involved processors, we have a highly parallel software, at hand. Finally, we showed the applicability of the algorithm to a physically relevant problem, including the (simplified) geometry of a butterfly, which is especially hard to resolve. The staggering amount of over 190 Million cells that we could use, shows the efficiency of our explicit update scheme, concerning the computing costs per time steps. The results for the computed drag and lift forces where plausible in all cases.

There are some limitations as well, however, that we don't want to conceal. The iterated transport of the level set function lead to severe problems with mass loss, at first. In the case of translations, we could overcome this problem by exploiting the structure of our grid, but we couldn't find a way to generalize this trick to rotations. Additionally, we saw that a Cartesian grid, despite its advantages, requires extremely many points to resolve any

really complex geometry, which might harm our accuracy, when calculating the forces, needed in a bidirectional coupling. All-in-all we can still be very pleased with our results.

Outlook

Despite the efforts undertaken, there are obviously many ways, in which the presented methods could be further improved. A fairly simple generalization, would be, to allow for multiple objects, moving independently of each other. Since the distance to a collection of objects, is the minimum over the distances to each individual object, the evaluation of the level-set function would be easy, even if each object was represented by it's own version of it. The only changes required would concern the infrastructure of the code.

Regarding more substantial extensions, there is the obvious generalization to a full fluid-structure interaction, by coupling the simulation to a structural FEM solver, which we set as a kind of far-term goal in the beginning of our project. Besides this, there is another improvement, that would require equally substantial effort to be realized. An adaptive grid, where regions around objects could receive a finer resolution, to spare some computational costs in the rest of the scene, would help us to reduce the needed amount of cells. Yet, to reap the full potential of this investment, we would also have to implement local time stepping, because otherwise the smallest cells in our scene would dictate the admissible time step over the whole domain, limiting the gain in efficiency. Together with the many times increased effort in handling the parallelization, there is quite some work to make an improvement along this path.

Acknowledgements

Obviously, the present thesis could not have been finished without the help and support of many people in the background. At this point I want to thank them all, for giving me the opportunity to work on this project, which proved to be very rewarding and teaching at the same time.

First of all I want to thank Prof. Dr. Griebel for handing me over this interesting topic and his great support in terms of giving ideas and outlining interesting directions. I also want to thank Prof. Dr. Schweitzer for taking over the role of the second assessor. On the same note I have to thank Markus Burkow, who introduced me to the NaSt3DGP fluid solver, took care of all of my day-to-day needs, and shared a lot of helpful insights along the way.

For providing me with all the infrastracture, especially the possibility to use the Atacama compute cluster, I thank the staff of the Institute of Numerical Simulation at the University of Bonn.

A major thanks goes to all members of the BRS-Motorsport Team at UAS Sankt Augustin, who played a major part in getting me interested in fluid problems, to begin with, and with whom I had two very special seasons of competing in the Formula Student, always paired with a lot of fun.

Additionally, I thank Sren Behr and my father Erwin for proof reading this thesis and giving valuable feedback on many points.

Finally, I want to thank my family for supporting me relentlessly, throughout my studies, and enabling all of this in the first place. Without the ongoing support of all of you nothing of this would be possible.

List of Figures

List of Tables

Literature

[Bat00] G. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge Mathematical Library. Cambridge University Press, 2000. ISBN: 978-0521663960.

[BM77] W. Briley and H. McDonald. 'Solution of the multidimensional compressible Navier-Stokes equations by a generalized implicit method'. In: *Journal of Computational Physics* 24.4 (1977), pp. 372–397. DOI: http://dx.doi.org/10.1016/0021-9991(77)90029-8.

[BMT12] E. Bayraktar, O. Mierka and S. Turek. 'Benchmark Computations of 3D Laminar Flow Around a Cylinder with CFX, OpenFOAM and FeatFlow'. In: *Int. J. Comput. Sci. Eng.* 7.3 (July 2012), pp. 253–266. DOI: http://dx.doi.org/10.1504/IJCSE.2012.048245.

[BW78] R. M. Beam and R. F. Warming. 'An Implicit Factored Scheme for the Compressible Navier-Stokes Equations'. In: *AIAA Journal* 16 (Apr. 1978), pp. 393–402. DOI: http://dx.doi.org/10.2514/3.60901).

[Cho68] A. J. Chorin. 'Numerical solution of the Navier-Stokes equations'. In: *Mathematics of Computation* 22 (1968), pp. 745–762. DOI: http://dx.doi.org/10.1090/S0025-5718-1968-0242392-2.

[Cro02] R. Croce. 'Ein paralleler, dreidimensionaler Navier-Stokes-Löser für inkompressible Zweiphasenströmungen mit Oberflächenspannung, Hindernissen und dynamischen Kontaktflächen'. MA thesis. Institut für Angewandte Mathematik, Universität Bonn, 2002.

[Cro10] R. Croce. 'Numerische Simulation der Interaktion von inkompressiblen Zweiphasenströmungen mit Starrkörpern in drei Raumdimensionen'. PhD thesis. Institut für Numerische Simulation, Universität Bonn, 2010.

[Enr+02] D. Enright et al. 'A Hybrid Particle Level Set Method for Improved Interface Capturing'. In: *Journal of Computational Physics* 183.1 (2002), pp. 83–116. DOI: http://dx.doi.org/10.1006/jcph.2002.7166.

[FP08] J. Ferziger and M. Perić. *Numerische Strömungsmechanik*. Springer Berlin Heidelberg, 2008. ISBN: 978-3540675860. DOI: http://dx.doi.org/10.1007/978-3-540-68228-8.

[GDN98] M. Griebel, T. Dornseifer and T. Neunhoeffer. *Numerical Simulation in Fluid Dynamics, a Practical Introduction*. SIAM, Philadelphia, 1998. ISBN: 978-3528067618.

[GGS82] U. Ghia, K. Ghia and C. Shin. 'High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method'. In: *Journal of Computational Physics* 48.3 (1982), pp. 387–411. DOI: http://dx.doi.org/10.1016/0021-9991(82)90058-4.

[Ghi+10] U. Ghia et al. 'The AIAA Code Verification Project - Test cases for CFD Code Verification'. In: *American Institute of Aeronautics & Astronautics AIAA* (2010). DOI: http://dx.doi.org/10.2514/6.2010-125.

[Gur81] M. E. Gurtin. *An Introduction to Continuum Mechanics*. Mathematics in Science and Engineering. Elsevier, 1981. ISBN: 978-0123097507.

[Hou+95] S. Hou et al. 'Simulation of Cavity Flow by the Lattice Boltzmann Method'. In: *Journal of Computational Physics* 118.2 (1995), pp. 329–347. DOI: http://dx.doi.org/10.1006/jcph.1995.1103.

[HS12] S. Haeri and J. Shrimpton. 'On the application of immersed boundary, fictitious domain and body-conformal mesh methods to many particle multiphase flows'. In: *International Journal of Multiphase Flow* 40 (2012), pp. 38–55. DOI: http://dx.doi.org/10.1016/j.ijmultiphaseflow.2011.12.002.

[KT00] A. Kurganov and E. Tadmor. 'New high-resolution central schemes for nonlinear conservation laws and convection-diffusion equations'. In: *Journal of Computational Physics* 160.1 (2000), pp. 214–282.

[Lee79] B. van Leer. 'Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method'. In: *Journal of Computational Physics* 32.1 (1979), pp. 101–136. DOI: http://dx.doi.org/10.1016/0021-9991(79)90145-1.

[LFO06] F. Lossaso, R. Fedkiw and S. Osher. 'Spatially Adaptive Techniques for Level Set Methods and Incompressible Flow'. In: *Computers & Fluids* 35.10 (Dec. 2006), pp. 995–1010. DOI: http://dx.doi.org/10.1016/j.compfluid.2005.01.006.

[LL87] L. Landau and E. Lifshitz. *Fluid Mechanics (Second Edition)*. Second Edition. Vol. 6. Course of Theoretical Physics. Pergamon, 1987. ISBN: 978-0080339337. DOI: http://dx.doi.org/10.1016/B978-0-08-033933-7.50001-5.

[LW60] P. Lax and B. Wendroff. 'Systems of conservation laws'. In: *Communications on Pure and Applied Mathematics* 13 (2 1960), pp. 217–237. DOI: http://dx.doi.org/10.1002/cpa.3160130205.

[Mac03] R. W. MacCormack. 'The Effect of Viscosity in Hypervelocity Impact Cratering'. In: *Journal of Spacecraft and Rockets* 40 (2003). Reprinted from AIAA Paper 69-354, 1969, pp. 757–763. DOI: http://dx.doi.org/10.2514/2.6901.

[Mac71] R. W. MacCormack. 'Numerical Solution of the Interaction of a Shock Wave with a Laminar Boundary Layer'. In: vol. 8. Proceedings of the Second International Conference on Numerical Methods in Fluid Dynamics. Springer-Verlag, 1971. DOI: http://dx.doi.org/10.1007/3-540-05407-3_24.

[Mac82] R. W. MacCormack. 'A Numerical Method for Solving the Equations of Compressible Viscous Flow'. In: *AIAA Journal* 20 (1982), pp. 1275–1281. DOI: http://dx.doi.org/10.2514/3.51188.

[Mac85] R. W. MacCormack. *Current Status of Numerical Solutions of the Navier-Stokes Equations*. AIAA 23rd Aerodpace Sciences Meeting. Jan. 1985. DOI: http://dx.doi.org/10.2514/6.1985-32.

[Mac93] R. W. MacCormack. 'A Perspective on a Quarter Century of CFD Research'. In: *AIAA Journal* 3291 (1993), p. 15. DOI: http://dx.doi.org/10.2514/6.1993-3291.

[OF01] S. Osher and R. P. Fedkiw. 'Level Set Methods: An Overview and Some Recent Results'. In: *Journal of Computational Physics* 169.2 (2001), pp. 463–502. DOI: http://dx.doi.org/10.1006/jcph.2000.6636.

[OF03] S. Osher and R. P. Fedkiw. *Level set methods and dynamic implicit surfaces*. Vol. 153. Springer Verlag, 2003. DOI: http://dx.doi.org/10.1007/3-540-05407-3_24.

[OS88] S. Osher and J. A. Sethian. 'Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations'. In: *Journal of Computational Physics* 79 (Nov. 1988), pp. 12–49. DOI: http://dx.doi.org/10.1016/0021-9991(88)90002-2.

[OT02] W. Oberkampf and T. Trucano. *Verification and Validation in Computational Fluid Dynamics*. Mar. 2002. DOI: http://dx.doi.org/10.1016/S0376-0421(02)00005-2.

[PH06] A. Perrin and H. H. Hu. 'An Explicit Finite-difference Scheme for Simulation of Moving Particles'. In: *Journal of Computational Physics* 212.1 (Feb. 2006), pp. 166–187. DOI: http://dx.doi.org/10.1007/1-4020-4977-3_17.

[SSO94] M. Sussman, P. Smereka and S. Osher. 'A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow'. In: *Journal of Computational Physics* 114.1 (1994), pp. 146–159. DOI: http://dx.doi.org/10.1006/jcph.1994.1155.

[THR76] J. Tannehill, T. L. Holst and J. V. Rakich. 'Numerical Computation of Two-Dimensional Viscous Blunt Body Flows with an Impinging Shock'. In: *AIAA Journal* 14.2 (1976), pp. 204–211. DOI: http://10.1016/j.jcp.2005.06.021.

[TS96] S. Turek and M. Schäfer. *Recent Benchmark Computations of Laminar Flow Around a Cylinder*. 1996. DOI: http://dx.doi.org/10.1007/978-3-322-89849-4_39.

[Wen08] J. Wendt. *Computational Fluid Dynamics: An Introduction*. A von Karman Institute book. Springer Berlin Heidelberg, 2008. ISBN: 978-3540850564.

[Wes01] P. Wesseling. *Principles of Computational Fluid Dynamics*. Lecture Notes in Computer Science. Springer, 2001. ISBN: 978-3540678533.

[Whi91] F. White. *Viscous Fluid Flow*. McGraw-Hill series in mechanical engineering. McGraw-Hill, 1991. ISBN: 978-0070697126.