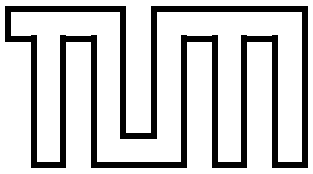


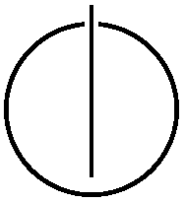
FAKULTÄT FÜR INFORMATIK

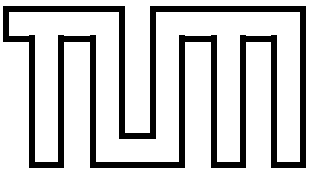
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Computer Science

Flexible Boundary Conditions for Fluid Solvers Based on Proximal Operators

Marie-Lena Eckert





FAKULTÄT FÜR INFORMATIK

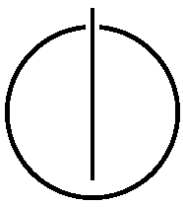
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Computer Science

Flexible Boundary Conditions for Fluid Solvers Based on Proximal Operators

Flexible Randbedingungen für Fluid-Löser Anhand von Proximalen Operatoren

Author:	Marie-Lena Eckert
Supervisor:	Prof. Dr. Nils Thuerey
Advisor:	Prof. Dr. Nils Thuerey
Date:	November 17, 2014



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, November 17, 2014 Marie-Lena Eckert

Acknowledgments

I would like to thank my supervisor, Professor Nils Thuerey, for his great support during the last year of my studies. Through his interesting research projects, I was able to develop even greater interest and passion for physically-based simulation. In many weekly and time-consuming meetings, I received aspiring guidance, great advice and friendly support from both Professor Nils Thuerey and Dr. Tiffany Inglis. I feel very lucky to have learned from some of the most talented and interesting people.
Furthermore, I would like to thank my family for sending me food supplies, warming messages, offering emotional remote support and supporting me financially throughout my whole studies.
However, studying computer science would be a whole different experience without my friends who always believe in me, support me no matter what and offer always good advice.
Last but not least, I want to thank Manuel Huber, who encourages me to follow my dreams, believes in me and brings the best out in me, often by being a great role model himself. This thesis would not be the same if it weren't for him.

Abstract

Due to the increasing use of physically-based simulation in the special-effects industry, it is important to provide fast and visually plausible fluid simulation. However, common fluid solvers feature sticky solid wall boundary conditions where the first layer of fluid is stuck to solid objects which in turn leads to fluid unnaturally crawling up walls and along ceilings.
In each fluid simulation, pressure equations must be solved in order to ensure boundary conditions and to make the fluid's velocity field divergence-free. When allowing only equality constraints for boundary conditions, the pressure equations form a system of linear equations. In order to allow the fluid to detach from solid objects, so called separating solid wall boundary condition must be enforced. Separating boundary conditions however feature normal velocity components at fluid-solid faces that are greater or equal to zero.
This inequality constraint can be solved by transforming the linear equation system into a Linear Complementarity Problem, which can only be solved by expensive Quadratic Programming solvers. Using expensive Quadratic Programming solvers stands in contrast to the desire for efficient fluid simulation.
The goal of this thesis is therefore to explore approaches for implementing separating and also flexible boundary conditions in general, while maintaining the system of linear equations in the pressure solve.
Recent findings show that it is possible to interpret the pressure solve as proximal operator. Proximal operators can in turn be integrated into proximal methods, which allow splitting difficult multi-objective optimization schemes into simpler sub-problems. This encourages modularity and the development of highly optimized solvers for each sub-problem. Such a proximal method is the Alternate Direction Method of Multipliers (ADMM), which has proven to be very efficient and successful in the field of statistics, machine learning and recently also in fluid simulation.
In this thesis, ADMM is implemented such that flexible boundary conditions are enabled and the only concerns in the pressure solve are the divergence-free velocity field and the free-surface boundary conditions. Another possibility to meet multiple conditions, such as non-divergence and boundary conditions, is the Iterated Orthogonal Projection (IOP). A series of projections is hereby applied iteratively. Hence, IOP is also implemented in order to achieve flexible boundary conditions and compared to ADMM. Furthermore, the possibility of integrating separating boundary conditions into a simple Conjugate Gradient (CG) pressure solve is investigated and implemented. To enhance the performance of ADMM and IOP, a hybrid method of the adjusted CG solver and IOP is developed. The results of all four techniques are compared and evaluated in detail.

Abbreviations

ADMM	Alternate Direction Method of Multipliers
AGP	Accelerated Global Preconditioning
BC	Boundary Condition
BFECC	Back and Forth Error Compensation and Correction
CFD	Computational Fluid Dynamics
CG	Conjugate Gradient
FFT	Fast Fourier Transformation
FLIP	Fluid-Implicit-Particle
IC	Incomplete Cholesky
IC(0)	Incomplete Cholesky Level Zero
IOP	Iterated Orthogonal Projection
LCP	Linear Complementarity Problem
MAC	Marked-and-Cell
MICCG(0)	Modified Incomplete Cholesky Conjugate Gradient Level Zero
NSEs	Navier-Stokes equations
PCG	Preconditioned Conjugate Gradient
PIC	Particle-in-Cell
PLS	Particle Level Set
QP	Quadratic Programming
SPH	Smoothed Particle Hydrodynamics

Contents

Chapter 1

Introduction

The research field of physically-based fluid simulation has gained a lot of attention starting from the late eighties [Bri08]. Fluids like water, smoke, dust, fire, explosions and related phenomena are breathtaking natural spectacles which cannot be omitted when it comes to model real world scenarios, e.g. in movies, video games or commercials. However, the desire for physically plausible animations exceeds the abilities of a human animator to explicitly specify the complex behavior of fluids interacting with objects and with other fluids frame by frame. In the early days, common fluid animation practices were simple approaches such as stop-motion, e.g. in "King Kong" in 1933, or keyframing, e.g. in "Gertie the Dinosaur" in 1912, and later the use of mass-spring networks, particle systems or bump map tricks.
Today, these practices are replaced by adapting advanced techniques from the field of Computational Fluid Dynamics (CFD). Fluid simulation has been modeled mathematically since the fifties and sixties due to its prime importance in most scientific disciplines and engineering. However, the aim of fluid simulation in computer graphics is the generation of plausible visual effects in contrast to predicting the correct flow fields in CFD [MSJT08]. With physically-based fluid simulation, animators are now able to almost effortlessly create interesting, swirling fluid-like behaviors including the interaction of flows with objects and virtual forces which in turn leads to lower production costs but unfortunately less direct fluid control.
Fluid simulation is as complex as it is fascinating due to the sophisticated mathematical fundamentals forming the backbone of all the playful movement of fluids. The equations governing a fluid's motion, the famous Navier-Stokes equations (NSEs), are unstable partial differential equations which are quite difficult to solve in an efficient way and even difficult to solve at all [GHD13]. Despite heavy simplifications of the NSEs, including the assumption of fluids to be incompressible, and the rapid growth of the computational power of CPUs and GPUs, fluid simulation would not be feasible for animation purposes without some major research breakthroughs in the past.
Harlow and his team paved the way for computer graphics when they developed the Particle-in-Cell (PIC) method in 1963 [Har63] and the Marked-and-Cell (MAC) grid structure in 1965 [HW65]. [FM96] were the first ones to use the MAC-grid approach in computer graphics to simulate fluids by approximating the NSEs on a staggered grid. Without Stam's significant work of the first unconditionally stable solver [Sta99], real-time interaction of physical fluid models would not be practicable as they are now in modern computer games. Due to its unconditional stability, the simulation time steps can be increased which leads to faster simulation but also to increased mass diffusion which diminishes the physical accuracy but not its profit for computer graphics.
Subsequently, hundreds of research publications lead to effective fluid solvers such that high-quality special effects are now commonplace [GB13]. While the special effects in "Waterworld" and "Titanic" in 1995 and 1997 were mostly limited to relatively calm and wide ocean shots, the current state of the art produces very realistic results so that people can hardly recognize the difference between real water splashing and the art of simulating it [Bri08]. Popular examples for the state of the art are close-up scenes of a stormy ocean interacting with a boat in the movie "The Wolf of Wall Street" (2013), vital flames and explosions in "Iron Man 3" (2013), crumbling stone statues burying everything underneath

them in "Immortals" (2011), droplet splashes in "Pirates of the Caribbean: On Stranger Tides" (2011) and the stunning, all-consuming flood in "2012" (2012), see Fig. 1.1.

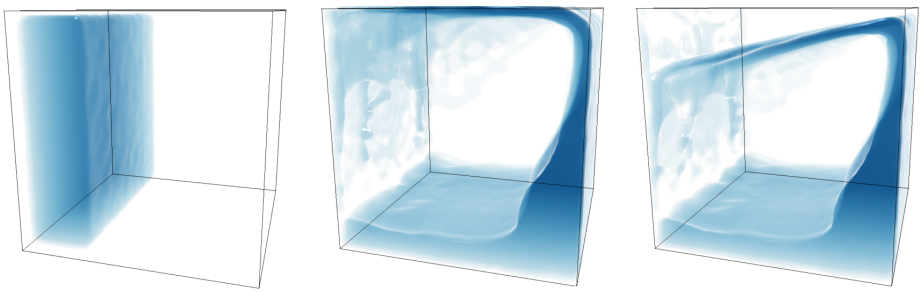
Figure 1.1: The Flood in the Movie "2012" by Scanline VFX [Sca14]

1.1 Motivation

Although subsequent research has refined, advanced and accelerated the pioneering work of Foster, Metaxas [FM96] and Stam [Sta99], current fluid solvers are still up for improvements in terms of unpredictable outcomes, numerical accuracy and performance. Especially the correct handling of boundary conditions turns out to be a quite difficult task [Bri08].
Boundary Conditions (BCs) arise wherever a volume of flowing material meets a volume of some other material [Lap03], e.g. the surface of contact between a liquid and air or its container such as a stream and its stream-bed.
The two essential parts in a fluid solver are on the one hand the advection of fluid quantities and on the other hand solving the pressure equations. The pressure's task is to ensure incompressibility and to hold the boundary conditions, i.e. preventing the fluid to flow into or out of solid walls, as explained Section 2.1.3. For the purpose of computer graphics, fluids are usually considered to be incompressible since compression rarely has a visual effect. Typically, the computational bottleneck of an incompressible fluid solver is the calculation of pressure at every time step, which requires solving a Poisson equation [MCPN08]. The most widely used method in the graphics community is the Preconditioned Conjugate Gradient (PCG) method with Incomplete Cholesky (IC) preconditioner to accelerate convergence. It is robust, easy to implement and can handle complex domain shapes. This method is an iterative technique for solving systems of linear equations and requires no explicit representation of the system's matrix. The drawback of PCG is that the runtime scales poorly for large grids and quickly becomes the CPU bottleneck.
Holding the solid boundary conditions means ensuring that the normal velocity component at fluid-solid faces is greater than or equal to zero. Unfortunately, introducing an inequality constraint turns the linear system of the discretized Poisson equation into a Linear Complementarity Problem (LCP) which is much more expensive to solve [CM11]. Restricting the normal velocity component of the liquid at fluid-solid faces to be zero leads to the crucial artifact of fluid crawling unnaturally along walls and ceilings and eventually dripping down or crossing over to another wall to descend [BBB07]. This numerical fact occurs

in free-surface simulation in current fluid solvers, e.g. in [CM11]. Fig. 1.2 (a) shows the initial state of a breaking dam scenario while (b) illustrates this very artifact. It impairs an animation's visual plausibility drastically. The desired fluid behavior is shown in Fig. 1.2 (c). It is provided by solving a LCP with a multigrid approach.

In nature, the phenomenon of liquids having zero normal velocity components at solid



(a) Initial Conditions (b) Sticky BC (c) Separating BC

Figure 1.2: 3D Breaking Dam Simulation by [CM11]

wall boundaries is indeed observed. A thin film of fluid is left on walls and ceilings, since the outermost molecules of the fluid are stuck to the surfaces past which it flows. However, this thin film is far too small to be resolved on an animation grid. Simulations using the 'no-slip' boundary condition, which means not slippery, enforce instead a layer of thickness unrealistically proportional to the grid cell size that sticks to the wall until some numerical error in advection eventually separates it [BBB07].

The issue of non-separating solid boundaries is targeted in [BBB07], [CM11] and [SB12] by applying expensive Quadratic Programming (QP) solvers, multigrid solvers and ghost Smoothed Particle Hydrodynamics (SPH). So far, neither have flexible boundary conditions yet been incorporated into a simple Conjugate Gradient (CG) solver nor was the handling of solid wall boundary conditions separated from the actual pressure solve. The problem of sticky solid boundary conditions is only one of many constraints regarding the boundary conditions in fluid solvers. By excluding the boundary conditions from the pressure solve, very efficient pressure solvers like the Fast Fourier Transformation (FFT), which can only handle the simplest boundary conditions, could be applied to solve the pressure equations while exhibiting great performance.

In [GITH14], the Alternate Direction Method of Multipliers (ADMM) is used to couple fluid simulation and fluid capture in order to improve fluid control. One of their key findings is that the pressure solve can be interpreted as a proximal operator, which enables its incorporation as a physical constraint into proximal methods. Proximal methods split difficult multi-objective optimization schemes into simpler sub-problems [PB13]. These sub-problems are expressed and accessed only by application-specific solvers known as proximal operators which encourages modularity and the development of highly optimized solvers for each sub-problem. ADMM is a thoroughly studied and elaborated technique which is supposed to converge very well for strictly convex problems when choosing a moderate accuracy. Therefore, ADMM could be applied in order to incorporate flexible solid wall boundary conditions into a fluid solver by solving the incompressibility condition independently from the solid wall boundary conditions.

In [MCPN08], an Iterated Orthogonal Projection (IOP) framework is used to enforce non-divergence and complex domain boundary conditions in terms of handling internal obstacles in a multigrid-based Poisson solver. IOP's suitability to enabling flexible boundary condi-

tions has to be investigated in this work, since both problems are quite similar.
Furthermore, it should be examined whether separating solid wall boundary conditions can be implemented within a simple CG solver when tolerating small deflections from the governing physical laws, e.g. introducing divergence at fluid-solid faces. On top of that, it is of importance to analyze and compare each method, namely ADMM, IOP and the adjusted CG solver.

1.2 Outline

Some of the related literature concern with creating separating solid boundary conditions but until now, no simple CG solver has been developed which enables fluid to detach naturally from solid walls or obstacles. Furthermore, solid wall boundary conditions are embedded in the common pressure solves which prohibits the application of complex boundary conditions. ADMM and IOP were proven to be suitable for and successful in fluid simulation applications.
Therefore, the overall goal of this work is to permit flexible boundary conditions by separating the boundary handling from the actual pressure solve with the application of ADMM and IOP. Furthermore, separating boundary conditions are incorporated into a simple CG solver by tolerating the creation of divergence at fluid-solid faces. To profit from the good performance of the adjusted CG solver as well as from the physical accuracy of the ADMM and IOP implementations, a hybrid method combining of IOP and the adjusted CG solver is developed.

The remainder of this thesis is organized as follows. In the next section, a detailed overview concerning the related work is provided. Basic knowledge about the fundamentals of fluid simulation, including the NSEs and CG, ADMM and IOP, is summarized in Chapter 2. Subsequently, the implementation of ADMM, IOP, the adjusted CG solver and the hybrid method is explained in detail in Chapter 3 with the use of code sketches. The choice of approaches is justified and their disadvantages are discussed. Furthermore, difficulties and issues to consider for the implementation of separating boundary conditions are illuminated. A thorough evaluation of the implemented techniques follows in Chapter 4, comparing their accuracy, convergence and performance as well as advantages and disadvantages. Each of the three methods is applied to several 2D and 3D scenes to examine their behavior and put their robustness up for a test. A conclusion of the findings as well as a discussion and an outlook are presented in Chapter 5.

1.3 Related Work

In this section, an overview about general literature concerning fluid simulation as an animation technique is provided in the first part. The second part covers literature about boundary conditions while the specific topic of sticky boundary conditions is examined in the end.
As mentioned before, [FM96] make the first approach to simulate fluids in computer graphics by solving the Navier-Stokes equations on a staggered grid [HW65] with a finite differences approach. They thereby introduce three-dimensional Eulerian liquid simulation and voxelize obstacles onto the grid for handling the solid BCs. Obstacles which are not aligned with the grid lead to significant stair-step artifacts which unfortunately do not converge to zero if the grid resolution is increased.

Stam [Sta99] provides the combination of implicit Poisson solvers and first-order semi-Lagrangian advection for faster simulation, which is widely used to visually simulate fluids today. [FF01] employs Stam's method and simulates liquids by tracking the surface with the Particle Level Set (PLS) method. Numerical errors and mass-loss are reduced by the introduction of Lagrangian marker particles. The fluid-solid coupling is handled by explicitly setting the normal velocity component to zero at fluid-solid faces and modifying the pressure solver to not change these velocities which mitigates the stair-case artifacts. The fluid is allowed to slip tangentially. For minimizing the artifacts of numerical dissipation, [FSJ01] introduces higher-order interpolation for advection as well as vorticity confinement. A different approach is proposed by [MCG03], where a mesh-free Lagrangian method is developed based on SPH. It is an extension of the Smoothed Particle Hydrodynamics (SPH)-based technique by [DG96] to animate highly deformable bodies. The fluid is represented by material particles with relatively simple interactions via sums of smooth kernel functions and their gradients. In contrast to Eulerian grid-based approaches, mass conservation and convection are easier to handle.
Back to grid-based approaches, [LGF04] provides enhanced simulation resolution by the usage of dynamically adapted octrees. Another advection improvement, the Back and Forth Error Compensation and Correction (BFECC), is presented by [KLLR05] to maximize the details resolved on a grid. [ZB05] introduces the incompressible Fluid-Implicit-Particle (FLIP) method to computer graphics. The FLIP method and its variants achieve a near total lack of numerical diffusion in the transport stage of the fluid simulation, since all quantities are advected on particles as opposed to a grid. It traces its history back to the early PIC work of [Har63] and MAC of [HW65]. [SFK+08] in turn presents a different advection technique, the MacCormack advection, and thereby addresses the loss of liquid mass and momentum. Another attempt to tackle the loss of details resolved on a grid due to dissipation is introduced in [KTJG08] by efficiently resolving frequencies greater than the Nyquist limit.
[Sta01] showed in 2001 that it is advantageous to take some steps of the numerical flow simulation in the frequency domain. While employing a semi-Lagrangian advection scheme in the spatial domain, he switches to the frequency domain to perform the pressure solve which leads to an order of magnitude higher performance. The simulation data is then transformed into the spatial domain for the next iteration. [Lap03] conducts a Navier-Stokes fluid flow simulation that operates entirely in the frequency domain to eliminate the costs of transformation between spatial and frequency domain. Simple optimizations such as common subexpression elimination and rearranging the loops for increased cache efficiency improve the simulation performance by a few percent. Restricting the grid dimensions to powers of two brings additional optimization for a total performance increase up to 20%. Also [Hen12] takes advantage of a FFT-based elliptic solver whose performance and scalability is optimal on shared-memory multiprocessors and is dramatically faster than the best iterative methods. This is a direct method that takes advantage of the fact that the Helmholtz equation can be solved independently for each Fourier mode as stated in [Hen12].

As a fluid is generally simulated in a domain with fixed and moving obstacles, it is necessary to consider the interaction of the fluid with these obstacles. Various research has addressed the coupling of solids and fluids in the physics, mathematics and computer graphics literature. First of all, a solid boundary condition, such that no liquid is allowed to enter or come out of the solid, must be ensured. Instead of setting all fluid velocities equal to the velocity of the object, [FF01] make the first improvement by allowing the fluid to move freely along the tangent of the solids. [NGF02] proceeds by using the Ghost Fluid

method to couple compressible fluids and deformable solids.
The topic of rigid body simulation is related to the handling of solid boundaries. For example, it faces the challenge of separating two rigid bodies from contact. [GHD13] targets the interaction of deformable solids with incompressible fluid using mass-spring systems, attaching the solid with ad hoc damped springs to nearby fluid marker particles. Looking closer into the interaction of water with air, [TFK$^+$03] introduces volume-of-fluid algorithms for animating multi-phase flow as opposed to simulating only the liquid with free-surface flows. Furthermore, they envisage solids which are not aligned with the grid and achieve solid-to-fluid coupling by setting the velocity of the fluid inside a cell containing a solid to the velocity of the solid. [REN$^+$04] propose object-liquid boundary conditions to allow natural interaction between the PLS representation of the liquid interface and the rigid bodies. Specialized boundary conditions are needed to ensure a visually pleasing interaction between the liquid interface and immersed objects. A set of object boundary conditions for the velocity field and both the level set function and particles comprising the particle level set method are set up. These boundary conditions are used for both moving and stationary objects. [CMT04] added better coupling between fluid and rigid body simulations by using distributed Lagrange multipliers to ensure two-way coupling that generates realistic motion for both the solid objects and the fluid as they interact with one another. The rigid objects are treated as if they were made of fluid. The rigidity of such an object is maintained by identifying the region of the velocity field that is inside the object and constraining those velocities to conform with rigid body motion.
The approach of [SB12] achieves less diffusion through the introduction of a narrow layer of ghost particles in the surrounding air and solid. This technique alleviates particle clumping that results when particles do not have sufficient neighbors to reach their target density. They focus on SPH where the fluid is represented by material particles with relatively simple interactions via sums of smooth kernel functions and their gradients. [BTT09] proposes a novel boundary handling algorithm for particle-based fluids based on a predictor-corrector scheme for both velocity and position. Different slip conditions can be realized and non-penetration is enforced.

While it is physically correct to enforce zero normal velocity at solid boundaries, it leads to the artifact of fluid sticking unnaturally to solid obstacles. [BBB07] and [CM11] implement freely separating boundaries by allowing the normal velocity component to be greater than zero. This leads to solving the complementarity condition $0 \leq p \perp u \cdot \hat{n} \geq v_{solid} \cdot \hat{n}$, which states that the normal velocity component can be greater than zero if the pressure is zero. Zero pressure changes the fluid-solid face to a free-surface. On the other hand if the pressure is greater than zero, the normal velocity component must be set to zero. The fluid is at rest at the wall and the positive pressure is acting on the fluid to keep it at rest to rule out suction from keeping the fluid stuck. This complementarity condition allows the fluid to freely separate from solid walls, similar to rigid bodies separating from contact.
[BBB07] additionally propose the variational interpretation of the pressure equation as kinetic energy minimization. For meeting the inequality constraint, an expensive QP solver is applied. To improve the performance of [BBB07], [CM11] solve the LCP with an efficient multigrid-based solver while [Erl13] aims at faster numerical ways for solving LCPs as well. [GB13] use the PCG wrapped in an active-set method and enforce non-negative pressures in all cells near the liquid surface.

None of the aforementioned literature provides an efficient solution for separating solid boundaries with the popular CG solver or a way to handle boundaries independently of the

pressure solver.
[GITH14] explore the connection between fluid capture, simulation and proximal methods. The proximal operator constraining fluid velocities to be divergence-free is directly equivalent to the pressure-projection methods commonly used in incompressible flow solvers. Proximal methods split difficult multi-objective optimization schemes into simpler sub-problems [PB13]. These sub-problems are expressed and accessed only by application-specific solvers known as proximal operators. ADMM has emerged as a powerful technique for large-scale structured convex optimization problems, especially in image processing and machine learning [Boy11]. While it was introduced for optimization in the seventies, its origins can be traced back to techniques for solving elliptic and parabolic partial difference equations developed in the fifties. ADMM profits from the strong convergence properties of the method of multipliers and the decomposability property of dual ascent and performs best when a moderate accuracy is chosen.
[MCPN08] introduce the IOP framework to calculate pressure over the grid. A series of orthogonal projections ensures that both non-divergence and solid boundary conditions are satisfied simultaneously to a specified accuracy. Both operators, the incompressibility projection and the boundary handling, are applied iteratively. It allows a simple and highly efficient multigrid method to enforce non-divergence in combination with complex domain boundary conditions. The free-surface boundary conditions cannot be satisfied, because Dirichlet conditions on the pressure field do not translate into linear constraints on the fluid velocity field. IOP shows slow convergence if the combined matrix of incompressibility and boundary handling has eigenvalues close to one, which means that the individual subspaces are almost parallel. Rapid convergence cannot be guaranteed.
[SPDC10] states that a common choice for the pressure's Poisson equation is the IOP method which requires a series of solutions on the complete mesh. They propose a variant of the IOP, called Accelerated Global Preconditioning (AGP). It is based on using PCG, in contrast to stationary methods used in IOP. The main advantage of AGP over IOP is the acceleration versus stationary. AGP iterates only on pressure, IOP iterates on both pressure and velocity. Instead of computing an intermediate pressure field in order to enforce the divergence conditions, IOP directly iterates on the velocity field. IOP iteration shows a linear rate of convergence because it is a stationary method.

Some of the related literature concern with creating separating solid boundary conditions but until now, no simple CG solver has been developed which enables fluid to detach naturally from solid walls or obstacles. Furthermore, solid wall boundary conditions are still embedded in the common pressure solves which prohibits the application of complex boundary conditions.

Chapter 2

Basics

In this chapter, the basics of fluid simulation including boundary conditions, numerical simulation, as well as ADMM and IOP are explained to build up a solid understanding for the problem of handling flexible boundary conditions based on proximal operators.

2.1 Fluid Simulation

Fluid simulations are not confined to just liquids. Fire and flames can also be produced by fluid simulations. When it comes to e.g. smoke, the fluid being simulated is the air itself which is a gas. The main difference between gases and liquids is that liquids have a free-surface and they are volume constrained as opposed to gases which do not unless dual or multi-phase simulation are strived for.
Simulating fluids for animation purposes implies solving the fundamental equations governing a fluid's motion: the famous incompressible Navier-Stokes equations (NSEs). The NSEs are a system of non-linear partial differential equations of second order. A simplified model of the NSEs are the Euler Equations, where viscosity is neglected. The Euler Equations are commonly used in the graphics community.
There are several approaches to solve those equations. The most common are Eulerian grid-based methods, Smoothed Particle Hydrodynamics (SPH) methods, vorticity-based methods and Lattice Boltzmann methods. The key difference in the graphic's setting to CFD is that the main feature of the results is visual plausibility. That is, if a human observer is unable to identify whether a given animation is physically correct, the results are sufficient. However, in physics, engineering, or mathematics, more rigorous error metrics are necessary.
In this thesis, the software framework of Mantaflow [man14] is used for fluid simulation. It comes with an Eulerian simulation using MAC-grids, a PCG pressure solver, FLIP simulations for liquids and many more features which are not relevant for this work. The simulations are conducted with the Fluid-Implicit-Particle (FLIP)-method as advection technique. For rendering, a level set method is used where an implicit surface function $\phi(i, j, k)$ is applied. The surface is defined as the set of points x where $\phi(x) = 0$.

2.1.1 Incompressible Navier-Stokes Equations (NSEs)

In this section, the incompressible Navier-Stokes equations (NSEs) are discussed in detail. Thereby, the presented information is based on [Bri08] if not stated otherwise.
The incompressible NSEs, a set of partial differential equations, govern a fluid's motion and hold throughout the fluid. They are composed of the *momentum equation*, described in Equation 2.1, and the *incompressibility condition*, described in Equation 2.2.
The *momentum equation* in Equation 2.1 specifies how the fluid accelerates due to the forces acting on it and can therefore be expressed as Newton's equation $\vec{F} = m\vec{a}$ or $\frac{F}{m} = \vec{a}$. The term $\frac{F}{m}$ is defined in Equation 2.3 while $\vec{a}$ is deduced in Equation 2.4.
The fluid force F is composed of three forces: pressure force F_p, force due to viscosity F_v and body forces including gravitational force F_g.

$$\frac{\delta \vec{u}}{\delta t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \quad (2.1) \qquad \nabla \cdot \vec{u} = 0 \quad (2.2)$$

where:

$\vec{u}$	=	velocity	ρ	=	density	p	= pressure
ν	=	kinematic viscosity	$\vec{g}$	=	body forces,	e.g.	acceleration due to gravity

F_p arises due to high-pressure regions pushing on lower-pressure regions. Only the net force, i.e. the difference of pressure, matters. The imbalance of pressure is exactly the negative gradient of pressure $-\nabla p$, which points towards lower-pressure regions. To obtain the pressure force, the negative gradient of pressure must be integrated over the volume V, thus $F_p = -V \nabla p$. Dividing the pressure force through the mass m leads to $\frac{F_p}{m} = -\frac{1}{\rho} \nabla p$.
A viscous fluid tries to resist deformation. Therefore, the viscosity force tries to make particles move at average velocity of nearby particles which is equal to minimizing differences in velocity between nearby bits of fluid. The Laplacian operator $\nabla \cdot \nabla$, a differential operator, measures how much a quantity differs from the average around it. The force due to viscosity consists of the Laplacian of the velocity v times the dynamic viscosity coefficient η integrated over the volume V, thus $F_v = V \eta \nabla \cdot \nabla u$. Dividing F_v through the mass m leads to $\frac{F_v}{m} = \nu \nabla \cdot \nabla u$, since $\nu = \frac{\eta}{\rho}$.
The gravitational force F_g is mass m times gravity $\vec{g}$ which henceforth leads to Equation 2.3.

$$\frac{F}{m} = \frac{F_p}{m} + \frac{F_v}{m} + \frac{F_g}{m} = -\frac{1}{\rho} \nabla p + \nu \nabla \cdot \nabla \vec{u} + \vec{g} \quad (2.3)$$

The acceleration $\vec{a}$ is exactly the material derivative $\frac{D\vec{u}}{Dt}$. Connecting the Lagrangian and the Eulerian viewpoint leads to an equation for the material derivative.
In the Lagrangian viewpoint, the continuum is treated like a particle system, where each particle has a position $\vec{x}$ and a velocity $\vec{u}$ and is tracked through time. When it comes to the Eulerian viewpoint, fixed points in the space are observed over time that contain measurements of fluid quantities like the velocity and density. This allows easier analytical work with spatial derivatives like pressure gradient than on a cloud of arbitrarily moving particles.
The Eulerian function $q(t, \vec{x})$ specifies the value of q at a time t for the particle that is at position $\vec{x}$. The Lagrangian question, how fast a quantity q is changing for the particle at position $\vec{x}$, is answered by taking the total derivative $\frac{d}{dt} q(t, \vec{x}) = \frac{\delta q}{\delta t} + \nabla q \cdot \vec{u}$. $\frac{\delta q}{\delta t}$ measures how fast the quantity q changes at a fixed point in space and $\nabla q \cdot \vec{u}$ defines how much of that change is just due to differences in the fluid flowing past. The total derivative is exactly the material derivative, see Equation 2.4.

$$\vec{a} = \frac{Du}{Dt} = \frac{\delta u}{\delta t} + \nabla u \cdot \vec{u} \quad (2.4)$$

The second equation is called *incompressibility condition* and described in Equation 2.2. Real fluids do change their volume on a microscopic level which is beyond a human's usual perception and therefore irrelevant for animation. Hence, fluids are treated as incompressible and a constant volume where zero rate of change is assumed. This forms exactly the incompressibility condition and can be expressed as $\int \int \int_\Omega \nabla \cdot \vec{u} = 0$, which is only true for $\nabla \cdot \vec{u} = 0$. A vector field that satisfies the incompressibility condition is called divergence-free, which is very hard to maintain. The zero divergence is assured by the pressure. The

pressure can be derived by taking the divergence of both sides in the momentum equation as shown in Equation 2.5.

$$\nabla \cdot \frac{1}{\rho} \nabla p = \nabla \cdot (-\vec{u} \cdot \nabla \vec{u} + \vec{g} + \nu \nabla \cdot \nabla \vec{u}) \tag{2.5}$$

In most animations, viscosity plays a minor role and can thus be neglected. The majority of numerical methods for simulating fluids unavoidably introduce errors that can be physically reinterpreted as viscosity. One of the biggest challenges in computational fluid dynamics is minimizing this viscous error as much as possible. The Navier-Stokes equations without viscosity are called the Euler equations, see Equations 2.6 and 2.7.

$$\frac{\delta \vec{u}}{\delta t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} \tag{2.6}$$

$$\nabla \cdot \vec{u} = 0 \tag{2.7}$$

2.1.2 Boundary Conditions

There are two kinds of boundary conditions: solid walls and a free surface. Both conditions are ensured by pressure.
To make sure that the fluid does not flow into solids or out of them, the normal component of the fluid's velocity $\hat{n} \cdot u$ must match the normal component of the solid's velocity $\hat{n} \cdot u_{solid}$. Inviscid fluids can freely slip past tangential directions which is called the no-stick condition. In contrast, the no-slip condition for viscid fluids enforces zero tangential velocity in relation to the solid. To enforce the solid wall boundaries, $\vec{u} \cdot \hat{n}$ must be controlled which leads to specifying $\nabla p \cdot \hat{n} = \frac{\delta p}{\delta \hat{n}}$.
Since air is 700 times lighter than water and comparable fluids, it does not have a big impact on the fluid. Air is therefore not treated as a fluid but as a region with constant atmospheric pressure which can be set to zero, since only differences in pressure matter. The velocity v is not controlled in any particular way. The free-surface boundary condition also applies for regions where the simulation is not further tracked. Because surface tension is only important for smaller-scale liquids, it is not considered here.

2.1.3 Numerical Simulation

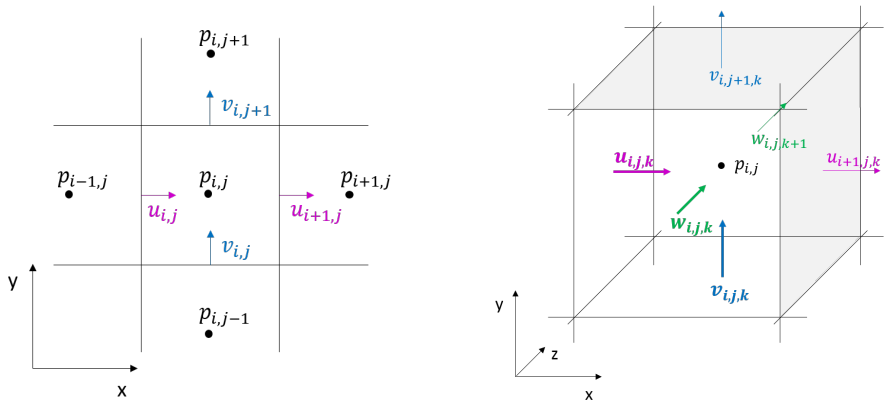
The basic equations for fluid simulation must be discretized for simulating fluids numerically in the computer. While there are many approaches for this, a high-quality approach is presented that works very well for graphics and is based on splitting.
Splitting implies the separation of an equation into its component parts and to solve each one separately in turn. The fluid equations are split up into an advection, body forces and pressure part as following.

$$\begin{aligned} &advect(\vec{u}, \triangle t, q): && \frac{Dq}{Dt} = 0 \\ &body(\vec{u}, \triangle t, \vec{g}): && \vec{u} \leftarrow \vec{u} + \triangle t \vec{g} \\ &project(\triangle t, \vec{u}): && \frac{\delta \vec{u}}{\delta t} + \frac{1}{\rho} \nabla p = 0 \quad \text{s.t. } \nabla \cdot \vec{u} = 0 \end{aligned} \tag{2.8}$$

$advect(\vec{u}, \triangle t, q)$ advects the quantity q through the velocity field $\vec{u}$ for a time interval $\triangle t$. For the body force $body(\vec{u}, \triangle t, \vec{g})$, forward Euler is used to update the velocity field $\vec{u}$. $project(\triangle t, \vec{u})$ calculates and applies the right pressure to make $\vec{u}$ divergence-free and also

enforce boundary conditions.
It is important that $advect(\vec{u}, \triangle t, q)$ is only performed in a divergence-free velocity field which must be ensured by $project(\triangle t, \vec{u})$. Therefore, the steps are applied sequentially.

Discretization of the Simulation Area The Marked-and-Cell (MAC) [HW65] is a new grid structure, also called staggered grid. The variables are stored at different locations in order to use accurate central differences to calculate the pressure gradient and the divergence of the velocity field without the usual disadvantages of the central differences. Although this structure is perfect for handling pressure and incompressibility, it is not well suited for evaluating the full velocity vector, since the weights for each component must be calculated for interpolation [Bri08]. Fig. 2.1 shows the staggered grid in two and three dimensions.



(a) The two-dimensional MAC-Grid (b) The three-dimensional MAC-Grid

Figure 2.1: The MAC-Grid

Advection Advection, also called convection or transport, is the process of moving a quantity within the velocity field v. As mentioned beforehand, it is a crucial step of fluid animation.
Solving the advection equation $\frac{Dq}{Dt} = 0$ is performed in the corresponding numerical routine $q^{n+1} = advect(\vec{u}, \triangle t, q^n)$ which returns an approximation to the result of advecting q through the velocity field over time step $\triangle t$. If the quantity q is moving around but does not change in the Lagrangian viewpoint, the material derivative is set to 0.
To avoid the instability of forward Euler and the spatial discretization problems due to standard central differences, a simple and physically-motivated approach called semi-Lagrangian method is used for advection [Sta99]. To advect particles, the particle that remains on position $\vec{x}$ must be identified and its value of q is looked up. To do this, the starting point of the current particle is tracked back in time by racing backwards through the velocity field u. The old value of q at that old position is exactly the new value of q at the new position. In case the old position is not directly on one of the grid points, the value of q is interpolated from the old values on the grid. The old position of the current particle is estimated with forward Euler or higher-order Runge-Kutta methods. Since a Lagrangian approach is used for an Eulerian calculation, this advection scheme is called semi-Lagrangian. If the old position happens to be outside the fluid due to numerical errors, the quantity q is extrapolated to the nearest point on the boundary.
The semi-Lagrangian approach is unconditionally stable since all values stay in the same

range which means that the quantity q is bounded.
In each advection step, averaging operations are performed which smooth out or blur sharp features. This appearance is called dissipation. An error analysis reveals that the numerical error due to dissipation is exactly a viscosity-like term. Therefore, viscosity must not be considered directly in the Navier-Stokes Equations which leads to the Euler equations.

2.1.3.1 Pressure

The heart of a fluid simulation is solving the pressure and thereby making the fluid incompressible and simultaneously enforcing boundary conditions.
$project(\triangle t, \vec{u})$ subtracts off the pressure gradient from the intermediate velocity field $\vec{u}$, such that the result satisfies the incompressibility condition inside the fluid and the solid wall boundary conditions. This process is summed up in Equation 2.9.

$$\vec{u}^{n+1} = \vec{u} - \triangle t \frac{1}{\rho} \nabla p \,, \quad \text{s.t.} \quad \nabla \cdot \vec{u}^{n+1} = 0 \quad \text{and} \quad \vec{u}^{n+1} \cdot \hat{n} = \vec{u}_{solid} \cdot \hat{n} \tag{2.9}$$

The goal is to establish a system of linear equations for finding the pressure. To do so, some discretizations must be specified.

Pressure Update Equations Each velocity component that borders a grid cell containing fluid is updated by the pressure. The pressure in air is zero while the normal velocity component at fluid-solid faces is equal to the normal component of the solid's velocity. This leads to the following pressure update Equation 2.10 for one velocity component u,

$$u_{i,j}^{n+1} = u_{i,j,} - \triangle t \, \frac{1}{\rho} \, \frac{p_{i,j} - p_{i-1,j}}{\triangle x} \qquad \Rightarrow \qquad p_{i,j} = p_{i-1,j} + \frac{\rho \triangle x}{\triangle t}(u_{i,j} - u_{i,j}^{n}). \tag{2.10}$$

Discrete Divergence Divergence has to be calculated only for a grid cell that is marked as fluid. The discrete divergence is shown in Equation 2.11.

$$\nabla \cdot \vec{u} = \frac{\delta u}{\delta x} + \frac{\delta v}{\delta y} + \frac{\delta w}{\delta z} \approx \frac{u_{i+1,j,k} - u_{i,j,k}}{\delta x} + \frac{v_{i,j+1,k} - v_{i,j,k}}{\delta x} + \frac{w_{i,j,k+1} - w_{i,j,k}}{\delta x} \tag{2.11}$$

With the help of the staggered grid in Fig. 2.1, these equations can be understood quite intuitively.

Linear Equation System for Finding the Pressure Equation 2.10 and 2.11 deliver the velocity update by the pressure gradient and the estimation of the divergence, which allows now the calculation of the incompressibility. Substituting the pressure update formulas, Equation 2.10, into the divergence formula, Equation 2.11, and setting the divergence to zero results in a linear equation for each fluid grid cell with the pressures as unknowns in Equation 2.12.

$$\begin{aligned} 0 \;=\; \frac{1}{\triangle x} \Bigg[& \left(u_{i+1,j,k} - \triangle t \, \frac{1}{\rho} \, \frac{p_{i+1,j,k} - p_{i,j,k}}{\triangle x} \right) - \left(u_{i,j,k} - \triangle t \, \frac{1}{\rho} \, \frac{p_{i,j,k} - p_{i-1,j,k}}{\triangle x} \right) \\ & + \left(u_{i,j+1,k} - \triangle t \, \frac{1}{\rho} \, \frac{p_{i,j+1,k} - p_{i,j,k}}{\triangle x} \right) - \left(u_{i,j,k} - \triangle t \, \frac{1}{\rho} \, \frac{p_{i,j,k} - p_{i,j-1,k}}{\triangle x} \right) \\ & + \left(u_{i,j,k+1} - \triangle t \, \frac{1}{\rho} \, \frac{p_{i,j,k+1} - p_{i,j,k}}{\triangle x} \right) - \left(u_{i,j,k} - \triangle t \, \frac{1}{\rho} \, \frac{p_{i,j,k} - p_{i,j,k-1}}{\triangle x} \right) \Bigg] \end{aligned} \tag{2.12}$$

This large system of linear equations for the unknown pressure values can be written as

$$Ap = b$$

where A is a large coefficient matrix, p is a vector of all pressure unknowns and b is a vector of negative divergences in each fluid grid cell.
In A, each row corresponds to one fluid cell and the entries are pressure coefficients which are mostly zero except possibly for the seven entries (in 3D) corresponding to $p_{i,j,k}$ and its six neighbors. Since only the non-zero entries are stored, A is a sparse matrix. The coefficients for neighboring fluid cells are $\frac{-\triangle t}{\rho \triangle x^2}$ and the coefficient for $p_{i,j,k}$ is $n_{i,j,k}\frac{\triangle t}{\rho \triangle x^2}$ where $n_{i,j,k}$ denotes the number of fluid- or air-cell neighbors. A is not only sparse but also symmetric. The coefficient of $p_{i+1,j,k}$ of row (i,j,k) must be identical to the coefficient of $p_{i,j,k}$ of row $(i+1,j,k)$. If one of those two cells is not fluid, the coefficient is zero. Otherwise it is the same non-zero value. Thus, only half of the non-zero entries must be stored in A.

Conjugate Gradient Algorithm (CG) Since matrix A is a well known type of matrix, the conventional way to solve the equation system is the efficient and simple to implement algorithm Modified Incomplete Cholesky Conjugate Gradient Level Zero (MICCG(0)).
The Conjugate Gradient (CG) algorithm is an iterative method. It can be guaranteed to converge to the solution and each iteration involves only basic computations that are easy to code and can be conducted by using highly optimized standard libraries. But the larger the grid, the longer the CG takes to converge. Preconditioning the CG speeds it up which is then called the Preconditioned Conjugate Gradient (PCG). CG takes more iterations the more A differs from the identity matrix I. Therefore, CG should be able to solve the preconditioned equations $M\ Ap = Mb$, where M is approximately the inverse of A, very fast.
Since computing $M = A^{-1}$ is too expensive, one of the Incomplete Cholesky Level Zero (IC(0)) family is taken as preconditioner to approximate M efficiently.
The residual $r_i = b - Ap_i$ measures the remaining divergence and is used as stopping criterion by comparing the infinity norm of the residual with a tolerance number that is inversely proportional to the simulation's length. To avoid inexact floating-point arithmetic, that prevents the algorithm to converge fully, a maximum number of iterations, e.g. 100 should be defined.
A is henceforth called Laplace Matrix.

Project Routine The $project(\triangle t, \vec{u})$ routine consists of calculating the negative divergence b with modifications at solid wall boundaries, setting the entries of A, constructing the preconditioner, solving $Ap = b$ with MICCG(0) and computing the new velocities according to the pressure gradient update as a last step. The routine is called *project*, since a projection is a linear operator which means applying it several times does not change the outcome. If the pressure solve ensures incompressibility and the boundary conditions in one step, there will be nothing to change in a second step since incompressibility and boundary conditions are already met.

Separating Boundary Conditions A new formulation for separating boundaries for free-surface flow is presented in [BBB07]. It allows the fluid to separate naturally from solids avoiding the common numerical artifact of fluid crawling up walls and along ceilings. The source of this problem is the $u \cdot \hat{n} = v_{solid} \cdot \hat{n}$ boundary condition which states that fluid

cannot flow into or out of a solid. The boundary condition is changed to $u \cdot \hat{n} >= v_{solid} \cdot \hat{n}$ by applying the complementarity condition $0 \leq p \perp u \cdot \hat{n} \geq v_{solid} \cdot \hat{n}$. The condition $u \cdot \hat{n} > v_{solid} \cdot \hat{n}$ enforces $p = 0$ making the interface act like a free-surface. On the other hand, if $u \cdot \hat{n} = v_{solid} \cdot \hat{n}$ then the fluid is at rest at the wall and there must be a pressure $p > 0$ acting on the fluid to keep it at rest. Therefore, suction is ruled out which would keep the fluid stuck to the wall..

2.2 Alternating Direction Method of Multipliers (ADMM)

Many problems of recent interest in statistics and machine learning can be posed in the framework of convex optimization [WBAW12]. The Alternate Direction Method of Multipliers (ADMM) is well suited to distributed convex optimization. It is closely related to dual decomposition, the method of multipliers, proximal methods and many others. It is a simple and powerful algorithm and can be viewed as an attempt to blend the benefits of dual decomposition and augmented Lagrangian methods for constrained optimization.
The algorithm solves problems in the form of

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x - z = 0 \end{array} \tag{2.13}$$

where $f, g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ are closed proper convex functions. The ADMM to the problem in Equation 2.13, also known as Douglas-Rachford splitting, is

$$\begin{aligned} x^{k+1} &:= \mathbf{prox}_{\lambda f}(z^k - y^k) \\ z^{k+1} &:= \mathbf{prox}_{\lambda g}(x^{k+1} + y^k) \\ y^{k+1} &:= y^k + x^{k+1} - z^{k+1} \end{aligned}$$

where k is an iteration counter [PB13]. The difference from the general linear equality-constrained problem is that the variable x is split into two parts, called x and z, with the objective function separable across this splitting. The variable y is the dual variable update. x and z are the slack variables and updated in an alternating fashion which accounts for the alternating direction.

Convergence When applying ADMM, the residual $r^k = x^k - z^k$ converges to zero as $k \to \infty$. The dual variable y approaches an optimal value $y*$ while x^k and z^k do not need to converge to optimal values.
ADMM can show poor converges for high accuracy. But it often converges to modest accuracy within a few tens of iterations [WBAW12].
The ADMM iteration is said to have converged, when the maximum value of the residual r^k is lower than an absolute threshold ϵ_{abs} plus a relative threshold ϵ_{rel} times the maximum value of x and z: $max(r^k) \leq \epsilon_{abs} + \epsilon_{rel} \cdot max(x^k, z^k)$.
Applying ADMM to the pressure solves makes x and z two velocity fields u and v. The dual variable y is from now on called q. The first proximal operator $\mathbf{prox}_{\lambda f}$ is an operator that makes the velocity field u divergence-free. $\mathbf{prox}_{\lambda g}$ is the operator which ensures that the solid wall boundary conditions are met in v.

2.3 Iterated Orthogonal Projection (IOP)

[MCPN08] use IOP to calculate the pressure over a grid by applying a series of orthogonal projections concerning non-divergence and boundary conditions iteratively. IOP allows to use a simple and highly efficient multigrid method to enforce non-divergence in combination with complex domain boundary conditions.
Removing the divergent part of the flow can be viewed as an orthogonal projection which can be written as a matrix-vector multiplication $x_{ndiv} = P_{ndiv}x$. The boundary condition enforcement procedure can be expressed as $x_{bound} = P_{bound}x$. Combining both projections leads to

$$P_{int} = lim_{n\to\infty}(P_{ndiv}P_{bounds})^n = lim_{n\to\infty}(P_{iter}P_{ndiv})^n. \tag{2.14}$$

Both projections are iteratively applied. The IOP framework independently satisfies both constraints and converges to x_{int}. Since the eigenvalues of both P_{iter} and P_{bound} are zero or one, the eigenvalues of P_{iter} are also between zero and one which guarantees convergence of an iteration that consists of simple repeated multiplication with P_{iter}. The convergence rate of IOP can be quite slow if P_{iter} has eigenvalues close to one which means that the individual subspaces are almost parallel. Rapid convergence cannot be guaranteed, but in practice, a limited number of iterations suffices for a satisfactory visual result.
IOP can be stopped when either the change between the velocity field from the previous iteration and the current velocity is small enough or when the divergence is eliminated up to a specified threshold.

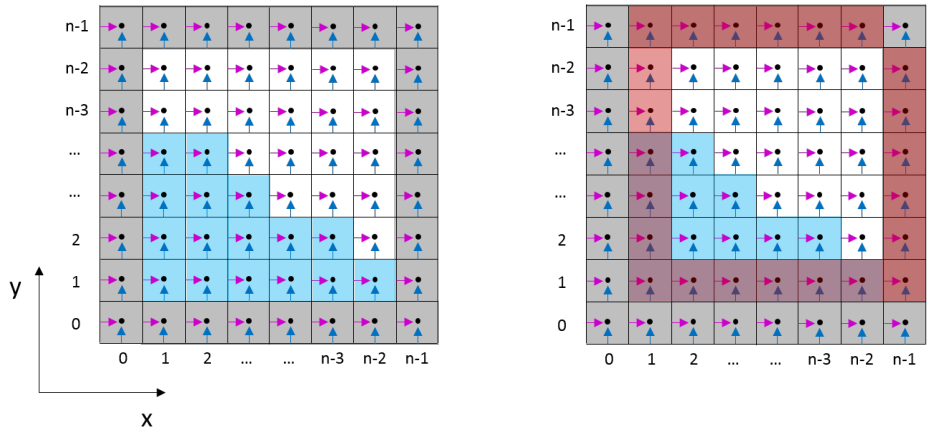
Chapter 3

Implementation

This chapter presents and explains the implementation of Alternate Direction Method of Multipliers (ADMM) and Iterated Orthogonal Projection (IOP) as pressure solve to allow for flexible solid wall boundary conditions in Section 3.2 and 3.3. Furthermore, difficulties in splitting the pressure solve properly into the incompressibility condition and boundary conditions are illuminated in Section 3.2. Inspired by the findings during the ADMM implementation, separating boundary conditions are incorporated into a simple CG solver in Section 3.4. Due to the rather poor performance of ADMM and IOP and physical inaccuracies in the adjusted CG solver, a hybrid method combining the advantages of each method is presented in Section 3.5.
But first of all, the original pressure solve as implemented in Mantaflow [man14] is introduced in Section 3.1. Mantaflow is the fluid simulation framework in which the pressure solve techniques are embedded. It is an open-source framework targeted at fluid simulation research in computer graphics. While the near-zero dissipation FLIP method is applied as advection scheme and the Preconditioned Conjugate Gradient (PCG) is used as pressure solve, particle level set methods based on [EF02] are used for surface tracking in order to the render the 3D simulations later on. Furthermore, it is assumed that obstacles and walls are not moving, i.e $u_{solid} = 0$.
The structure of a two-dimensional MAC-grid used in Mantaflow is shown in Fig. 3.1 (a). Additionally to velocity and pressure information, each cell is classified by flags as fluid, solid or air cell. Gray cells represent solid cells, blue is associated with fluid while air cells are transparent. Purple arrows symbolize velocity components into the x-direction, whereas blue arrows indicate y-velocity components. Black dots represent the pressure of each cell. In order to access the normal velocity components at fluid-solid faces, the underlying grid structure shown in Fig. 3.1 (a) must be considered. When iterating over the grid to modify those velocity components, it has to be determined from which cell they are accessed. It is assumed, that each lower velocity component, i.e. the left x-component and the bottom y-component, are accessed from the current cell. Therefore, when accessing velocity components at the left fluid-solid border, the very velocity components are accessed from fluid cells, as highlighted in red in Fig. 3.1 (b). In contrast, the velocity components at the right fluid-solid border are accessed from solid cells.

3.1 Original Conjugate Gradient Pressure Solve (OrigPS)

In this section, the original CG pressure solve implemented in Mantaflow is presented and explained. Thereby, the implementation of ADMM and IOP in Section 3.2 and 3.3 as well as the adjusted CG pressure solve and the hybrid method in Section 3.4 and 3.5 can be comprehended very well.
The original pressure solve in Mantaflow is henceforth abbreviated by **OrigPS**.
After advecting the fluid quantities and applying body forces at the beginning of each simulation step, the velocity field must be modified such that it is divergence-free and the boundary conditions must be ensured. Therefore, the pressure solve is conducted which implies solving $Ap = b$ for the pressure p. The pressure solve is encapsulated in the procedure *solvePressure* sketched in a simplified form in Algorithm 1.



(a) 2D MAC-Grid with Flags in Mantaflow (b) Cells Containing the Fluid-Solid Velocity

Figure 3.1: The 2D MAC-Grid in Mantaflow

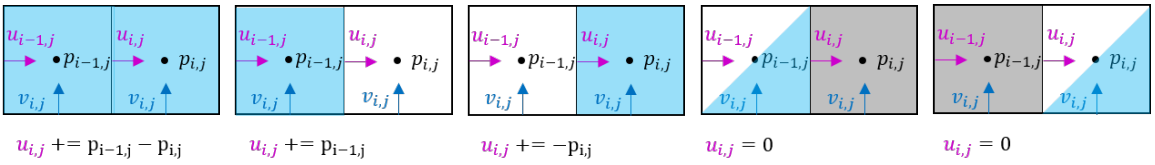
First, the Laplace Matrix A is generated in *makeLaplaceMatrix* in line 2 according to Section 2.1.3. Each row in A corresponds to one fluid cell and the entries are the pressure coefficients which are mostly zero except possibly for the seven entries (in 3D) for $p_{i,j,k}$ and its six neighbors. For building A, information about the cell classification must be provided in the variable *flags*. In the original pressure solve, the variable *flags* contains the unmodified cell classification as shown in Fig. 3.1 (a).
In line 3, the right-hand side, vector b, of the equation system is formulated by setting it to the divergence of each fluid cell. To do so, the method *makeRhs* is called. To calculate the divergence of each fluid cell, the velocity field u and the cell classification *flags* must be provided. Then, the linear equation system is solved iteratively by the Preconditioned

Algorithm 1 Solve p in $Ap = b$ and Update u

```
1: procedure SOLVEPRESSURE(u,p,flags)
2:     makeLaplaceMatrix(A, flags)
3:     makeRhs(b, u, flags)
4:     for i ← 1 to maxIterations do
5:         gcg → iterate(A, p, b)
6:     end for
7:     correctVelocity(u, p, flags)
8: end procedure
```

Conjugate Gradient (PCG) in line 5. If the solution for p is approached sufficiently, i.e. the residual is small enough, the iteration variable i is set to *maxIterations* which leads to breaking out of the loop.
As a last step, the velocity field u is updated with the negative pressure gradient calculated from p in *correctVelocity*, see line 7. Therefore, the velocity field now holds both the incompressibility condition and the boundary conditions as explained in Equation 2.10 in Section 2.1.3.
The pressure solve assigns pressure values to fluid cells for correcting the velocity components adjacent to fluid cells. The pressure in air cells is assumed to be zero while pressure in fluid cells is assumed to be equal to the neighboring fluid cell's pressure, such that the pressure gradient is zero. Therefore, the velocity at fluid-solid or air-solid faces is never

updated. The rules governing the pressure update applied in *correctVelocity* are visualized exemplary for x-velocity components in Fig. 3.2. Between two fluid cells, the negative



(a) Fluid-Fluid (b) Fluid-Air (c) Air-Fluid (d) Fluid/Air-Solid (e) Solid-Fluid/Air

Figure 3.2: Rules for Updating the Velocity in the original Pressure Solve

pressure gradient is added to the existing velocity, see Fig. 3.2 (a). Since the pressure in air cells is assumed to be zero, the pressure gradient updating velocity components between air and fluid cells is reduced to the pressure value of the corresponding fluid cell, see Fig. 3.2 (b) and (c). The pressure gradient at faces adjacent to solid grid cells is assumed to be zero. Therefore, the velocity at solid cell faces is always set to zero as shown in Fig. 3.2 (d) and (e).
Note that it is now important to bare in mind, to which cell the velocity component between two cells belongs to. As aforementioned, the velocity component $u_{i,j}$ belongs to the right cell in each image in Fig. 3.2.
Unfortunately, the implementation of the original pressure solver leads to the disturbing artifact where fluid is crawling upwards on walls and sticking to ceilings. To avoid this, the normal velocity components at fluid-solid faces cannot be set to zero. In order to allow these very velocity components to be greater or equal to zero, the pressure solve is modified in four different approaches as explained in Section 3.2, 3.3, 3.4 and 3.5.

3.2 ADMM as Pressure Solve (ADMMSep or ADMMStick)

In this section, ADMM is applied as pressure solve in order to allow for flexible solid wall boundary conditions. This method is henceforth abbreviated with **ADMMSep** when separating boundary conditions are simulated and **ADMMStick** in case sticky boundary conditions are applied.
To generate a velocity field that is both incompressible and meets the boundary conditions with ADMM, two proximal operators must be implemented, one for each sub-problem. It turns out to be quite difficult to split the pressure solve properly into two distinct sub-problems. Furthermore, it is not possible to use pressure as slack variables since the pressure values are never read. Therefore, ADMM has to operate on two velocity fields u and v. The first proximal operator is the original pressure solve applied with a modified cell classification. Each solid cell is now classified as air cell. Therefore, only the incompressibility condition and the free-surface boundary conditions are handled in the pressure solve. This first proximal operator is implemented in the procedure *solvePressure* as presented in Section 3.1. The second proximal operator ensures the solid wall boundary conditions and is encapsulated in the procedure *ensureBC*.
The algorithm of using ADMM as pressure solve is sketched in Algorithm 2. In line 2, the second velocity slack variable, namely v, is initialized with a copy of u. The variable q is the dual update variable and hence a velocity field. It is initialized with zero in line 3. The residual $r = u - v$ is a velocity field as well and initialized with zero in line 3. Since *solvePressure* has to operate on a modified cell classification where solid cells are replaced

Algorithm 2 Solve $Ap = b$ by means of ADMM

```
1: procedure APPLYADMM(flags,u,p,ϵ_CG,ϵ_Aabs,ϵ_Arel)
2:     v = copy(u)
3:     q,r = MACGrid(0)
4:     noSolidsFlags = removeSolids(flags)
5:     for i ← 1 to maxIterADMM do
6:         solvePressure(u,p,noSolidsFlags,ϵ_CG)
7:         if i==1 then markDetachingCells(u,p,flags) end if
8:         v = u + q
9:         ensureBC(flags,v)
10:        q = q + u - v
11:        r = u - v
12:        if max(r) <= ϵ_Aabs + ϵ_Arel * (max(u, v)) then break end if
13:        u = v - q
14:    end for
15: end procedure
```

by air cells, the variable *noSolidsFlags* is instantiated in line 4. It is a copy of the original cell classification where solid cells are replaced by air cells.
In the ADMM loop starting in line 5, the first proximal operator, *solvePressure*, is applied in line 6. After this, the variable v must be updated in line 8. The procedure *markDetachingCells* is described later. In line 9, the second operator is applied to v and ensures the solid wall boundary conditions. To do so, it is iterated over the grid and the velocity components at fluid-solid faces less than zero are set to zero. Only velocity components greater or equal to zero are allowed. In Section 3.1, it is explained how to iterate over the MAC-grid in order to modify the velocity components. After the application of the second proximal operator, the dual variable is updated in line 10. Furthermore, the residual r is calculated in line 11 to check for the stopping criterion in line 12.

Convergence In line 12, the ADMM iteration is stopped when convergence is reached according to Section 2.2. Common values of ϵ_{Arel} range between 10^{-3} and 10^{-4}. For ϵ_{Aabs}, a value on the scale of typical variable values is chosen which is here 10^{-3} as well. When using the stopping criterion based on $r_{vel} = u - v$, it is taken into account how well both problems of incompressibility and boundary conditions are solved simultaneously. But it is also possible to calculate the divergence of u and stop the iteration when $r_{div} = div(u) \leq \epsilon_{Adiv}$. A practical value for ϵ_{Adiv} is 10^{-2}. Note that the divergence should be calculated *after* the dual update of u in order to measure the divergence of the velocity field where both proximal operators are combined.
In order to compare OrigPS, ADMM and IOP later on, *solvePressure* in Algorithm1 is modified to stop when the divergence of the resulting velocity field is below the threshold ϵ_{CG}. To do so, *correctVelocity* is applied to a temporary velocity field after each CG iteration. When the divergence of the temporary velocity field is low enough, the iteration is stopped.
For better convenience, the kind of residual used in the method is added to its name, e.g. ADMMVelSep or IOPDivSep.

Classifying Solid Cells According to their Role in the Fluid-Solid Interaction In order to allow the fluid to detach from solid objects, each solid cell must be classified as free-surface

or solid wall depending on its current role. If the fluid is flowing towards a wall, solid cells act as a wall to withstand the fluid's pressure and to ensure that the fluid does not flow into the solid. The pressure inside the solid cell is then assumed to be equal to the pressure in the adjacent fluid cell so that the fluid cannot flow into the solid. On the other hand, if the fluid is not pushing against the wall, the boundary acts like a free-surface. This is the case for phases where the fluid detaches from the wall e.g. at the ceiling. To allow the fluid to detach from the wall, the solid cells are then treated as air cells by considering the fluid cell's pressure in the pressure update.
The method *markDetachingCells* is from crucial importance. Positive normal velocity at fluid-solid faces are allowed in *ensureBC* when separating boundary conditions are desired. But when the normal velocity at a solid cell c is set to zero to prevent fluid to flow into the solid cell in iteration $i = m$, and that very normal velocity is allowed to be greater than zero in another iteration $i = n$, the resulting fluid simulation exhibits incorrect fluid behavior. The fluid is getting pushed away from solid walls for one cell width and is jittering in domain corners. This results from treating a fluid-solid face differently in each iteration. The classification of solid cells into free-surface and solid walls must be established before the first application of *ensureBC* and must not be changed during one time step. The solid cells are classified as free-surface if the normal velocity component at its face to the neighboring fluid cell is greater than zero.
For this reason, the method *markDetachingCells* is developed as shown in Algorithm 3. To ensure the solid boundary conditions, it is iterated over the whole grid. For each grid

Algorithm 3 Classify Solid Cells into Free-Surface and Solid Wall

```
1: procedure markDetachingCells(u,flags)
2:     for each cell c do
3:         if isFluid(c) then
4:             if isSolid(left) AND u.x > 0 then setAir(left) end if
5:             if isSolid(bottom) AND u.y > 0 then setAir(bottom) end if
6:             if isSolid(front) AND u.z > 0 then setAir(front) end if
7:         else if isSolid(c) then
8:             if isFluid(left) AND u.x < 0 then setAir(c) end if
9:             if isFluid(bottom) AND u.y < 0 then setAir(c) end if
10:            if isFluid(front) AND u.z < 0 then setAir(c) end if
11:        end if
12:    end for
13: end procedure
```

cell, the left, lower and front neighbors are evaluated. Therefore, each fluid-solid face is captured. If the current cell is a fluid and one of the neighbors is a solid cell, the normal velocity component at this very fluid-solid face is evaluated. If the velocity component in relation to the normal direction is positive, this solid cell is classified as free-surface. For convenience, this cell is marked as air in the *flags* variable. Note that the flags variable provided to *applyADMM* should be a copy of the original variable such that the rest of the fluid simulation is operating on the original *flags* variable again.
The procedure *ensureBC* is not aware of any classification but sets each velocity component at fluid-solid faces to zero. Solid cells which were modified to air by *markDetachingCells* are therefore not changed at all. Note that the fluid at solid cells classified as free-surface is never flowing into the wall although this could be possible when the value in the neighboring fluid cell is changing to a positive value during the ADMM iteration.

In order to compare ADMM with the original pressure solver OrigPS, ADMM can also be applied with sticky walls. Then, no solid cell is classified as free-surface and the same sticky behavior is achieved.
Note that the classification of the solid cells can also be conducted by checking on the pressure values in the neighboring fluid cells. If the pressure is negative, the solid cell is classified as free-surface since the fluid is in the process of detaching from the wall.

3.3 IOP as Pressure Solve (IOPSep or IOPStick)

Since the Iterated Orthogonal Projection (IOP) has already been successfully applied to a similar problem in [MCPN08], IOP is now implemented to allow flexible solid wall boundary conditions in the pressure solve.
The implementation of IOP resembles the implementation of ADMM. However, it is less sophisticated. Applying IOP with sticky boundary conditions is from now on abbreviated with **IOPStick** and IOP with separating boundary conditions is called **IOPSep**.
Since both *solvePressure* and *ensureBC* operate on the same velocity field u when it comes to IOP, there is no need for the variables v and q. A similar stopping criterion to ADMM would be to compare the velocity field from the previous time step $u_{previous}$ to the current velocity field $u_{current}$. However, the residual $r_{vel} = u_{current} - u_{previous}$ accounts for the changes from one time step to another and not for the difference between the slack variables which accounts for how well both problems are solved. Since both operators are applied to the same velocity field, the second operator could change the velocity field again so that the total change is less than after the first operator. Simulation results show in case r_{vel} is used as residual and the same thresholds are used for ADMM and IOP, that IOP simulations feature semi-permeable walls while ADMM simulations require more iterations and produce correct results.
Therefore, the default stopping criterion for IOP is the divergence. It can be assumed that if the incompressibility is well solved, which can be measured in the divergence of u, the boundary conditions are solved similarly well since both problems are solved on one single variable u.

In Algorithm 4, the code snipped for applying IOP as pressure solve is shown. Besides the different use of the velocity field variables, all method calls are the same. In line 2, the residual r is initialized while the modified cell classification is instantiated in line 3. The IOP loop starts in line 4. The pressure projection is applied in *solvePressure* in line 5, followed by the updating the classification of the solid cells in line 6. The projection

Algorithm 4 Solve $Ap = b$ by means of IOP

```
1: procedure APPLYIOP(flags,u,p,ϵ_CG,ϵ_Idiv)
2:     r = MACGrid(0)
3:     noSolidsFlags = removeSolids(flags)
4:     for i ← 1 to maxIterIOP do
5:         solvePressure(u,p,noSolidsFlags,ϵ_CG)
6:         if i==1 then markDetachingCells(u,p,flags) end if
7:         ensureBC(flags,u)
8:         r = div(u)
9:         if max(r) <= ϵ_Idiv then break end if
10:    end for
11: end procedure
```

to ensure the boundary conditions *ensureBC* is applied on u in line 7. The divergence as residual is calculated in line 8 and evaluated in line 9.

3.4 Conjugate Gradient Pressure Solve with Separating Boundaries (OrigPSSep)

Inspired by the idea to allow positive normal velocity components at solid-fluid faces, the original conjugate gradient pressure solve is extended to implement separating solid wall boundary conditions as well. This implementation is in the following referred to as **OrigPSSep**.

The only adjustment in Algorithm 5 compared to Algorithm 1 is the cell classification in line 7. After solving $Ap = b$ iteratively in Algorithm 1, the method *markDetachingCells* is called before the pressure update is conducted in *correctVelocity* as shown in Algorithm 5. The solid cells are either classified by means of the velocity or by checking for negative pressure in the adjacent fluid cells. When using the classification based on the velocity, the pressure update must be conducted on a temporary velocity field where all solid cells are changed to air cells. Those solid cells that possess a positive normal velocity component at a fluid-solid face after the update are then classified as free-surface, i.e. set as air cell for the actual update in *correctVelocity*. Classifying solid cells as free-surface if their adjacent fluid cell features negative pressure leads to separating boundary conditions as well.

Since the pressure solve is conducted using a different classification of solid cells than for

Algorithm 5 Solve p in $Ap = b$ and Update u with Separating BCs

```
1: procedure solvePressure(u,p,flags)
2:     makeLaplaceMatrix(A, flags)
3:     makeRhs(b, u, flags)
4:     for i ← 1 to maxIterations do
5:         gcg → iterate(A, p, b)
6:     end for
7:     markDetachingCells(u,p,flags)
8:     correctVelocity(u, p, flags)
9: end procedure
```

the pressure update, divergence is introduced at the fluid-solid faces where a free-surface is determined. While this does not lead to visual artifacts for most scenarios, it is still an objectionable result since the advection step should only be applied on a divergence-free velocity field.

Classifying the solid cells before the pressure solve, directly after the application of body forces, leads to semi-permeable walls due to a high number of solid cells classified as free-surface. Nevertheless, a solution for eliminating the divergence is presented in the next section.

3.5 IOP with Additional Boundary Information (IOPSepBC)

Since ADMM and IOP might perform not fast enough for some practical application, the convergence is accelerated by supplying the pressure solve again with information about the solid boundary conditions. Obviously, the two sub-problems are then no longer separated. Furthermore, ADMM with r_{vel} can no longer be carried out. The velocity field returned

by *solvePressure* already meets all boundary conditions, so that *ensureBC* does not have any impact in v. Therefore, $r_{vel} = u - v$ is always zero and ADMM is equal to the original pressure solve.
Applying IOP with BCs information passed to the pressure solve is more suitable because the operation takes place on only one velocity field.
Nevertheless, applying ADMM or IOP with using r_{div} and supplying boundary information to the pressure solve is exactly applying OrigPSSep several times to rule out the introduced divergence.

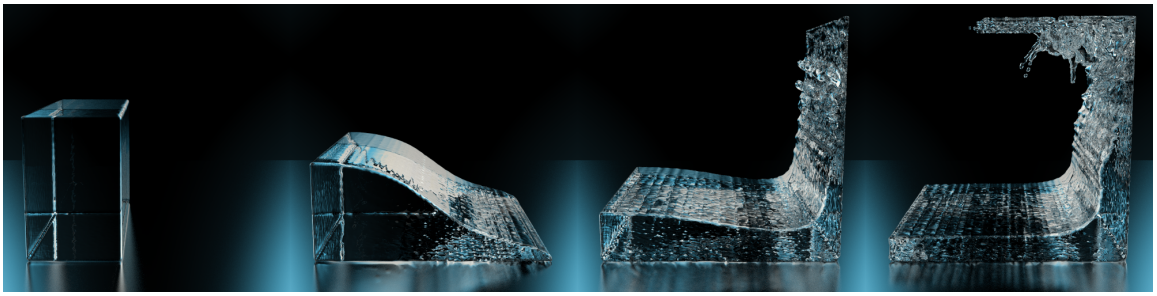
In this chapter, four different methods as pressure solve are introduced, additionally to the original pressure solve. In the following chapter, the five variants of the pressure solve, namely OrigPS, OrigPSSep, ADMMSep, IOPSep and IOPSepBC, are simulated and extensively evaluated.

Chapter 4

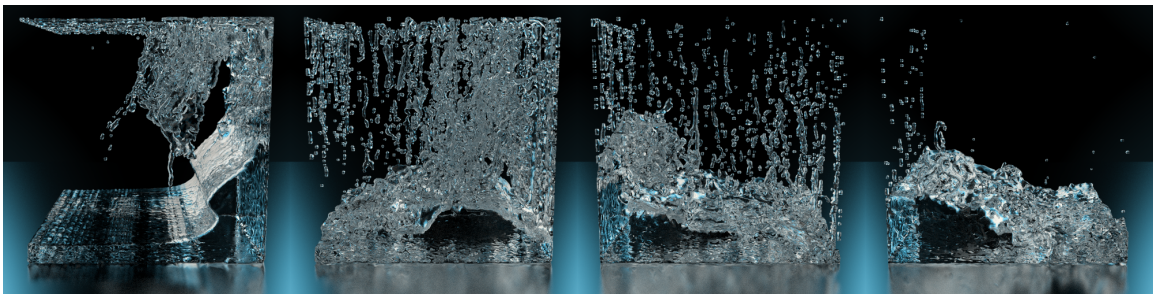
Results and Evaluation

In this part, the previously implemented methods called original pressure solve with separating walls (OrigPSSep), ADMM and IOP as pressure solve with separating boundary conditions (ADMMVelSep and IOPDivSep) are evaluated and compared. The resulting fluid simulations are therefore graphically demonstrated and analyzed.
This chapter starts with presenting a breaking dam scene simulated by OrigPS in 3D and moves on to reproducing the same scenario with OrigPSSep, ADMMVelSep and IOPDivSep. As a next step, the convergence of all methods and their variants is exposed and the choice of the stopping criterion is discussed in Section 4.1. The resulting performance of the methods is illustrated afterwards in Section 4.2 and leads to an investigation of how far the solutions differ from each other in Section 4.3. In Section 4.4, the hybrid method IOPDivBC is explained and evaluated. Section 4.5 compares the cell classifications based on velocity and pressure. As a last part, the methods are applied to different scenes to put their robustness and behavior with static obstacles to the test in Section 4.6.

Breaking Dam Scenario As a first evaluation step, all variants of the pressure solve are compared to each other by simulating a breaking dam scene as follows. A fluid box is placed on the left side of a squared domain. Fig. 4.1 shows that during time, the fluid breaks down, splashes against the right wall at $t = 37$, (b), crawls up the right wall, hits the ceiling at $t = 66$, (c), and reaches the left wall at $t = 120$, (e), and breaks down again until the main fluid body hits the left wall at $t = 160$, (f), and forms a big wave pressing against it at $t = 200$, (g). The simulation is run for 250 steps, i.e. $t_{max} = 250$. The time step δt is 0.5 for all simulations and the three-dimensional grid size is $96 \times 96 \times 96$. The default accuracy for the CG solver is $\epsilon_{CG} = 10^{-3}$. Unfortunately, the fluid is sticking unnaturally to the ceiling and does not detach until $t = 200$. Even after $t = 200$, a few fluid drops glue to the ceiling. This incorrect behavior fuels exactly the motivation for exploring other methods which cause the fluid to detach naturally from solids, e.g. walls, obstacles and ceilings.
The breaking dam scenario is now simulated with OrigPSSep, ADMMVelSep and IOPDivSep as shown in Fig. 4.2. For improving the distinction between these, the 3D simulations are rendered in different colors, i.e. blue, green, pink and yellow for OrigPS, OrigPSSep, ADMMVelSep and IOPDivSep. The accuracy for the CG solver inside ADMM and IOP, ϵ_{CG}, is set to 10^{-3} as well. IOP uses the divergence as stopping criterion, while ADMM stops on behalf of how close the two different velocity fields u and v are, as explained in Section 4.1. The accuracy ϵ_{Idiv} is set to 10^{-2}, ϵ_{Aabs} and ϵ_{Arel} to 10^{-3}. These configurations of IOP and ADMM produce very similar results which makes them well comparable. Note that the subscript 'CG' stands for the accuracy of the CG pressure solve in every method, while 'I' stands for IOP and 'A' for ADMM. When using the velocity as residual in ADMM, two accuracies need to be specified, i.e. an absolute and a relative value as explained in Section 2.2.
In opposite to the results produced by OrigPS in Fig. 4.2 (a)-(d), the simulations with OrigPSSep, ADMMVelSep and IOPDivSep feature well detaching fluid behavior since all three methods allow positive normal velocity components as shown in Fig. 4.2 (c)-(p). At $t = 120$, a thin fluid layer is still crawling along the ceiling for OrigPS in Fig. 4.2 (c) while the fluid is already completely detached at $t = 100$ when using separating boundary condi-



(a) $t = 0$ (b) $t = 37$ (c) $t = 66$ (d) $t = 100$



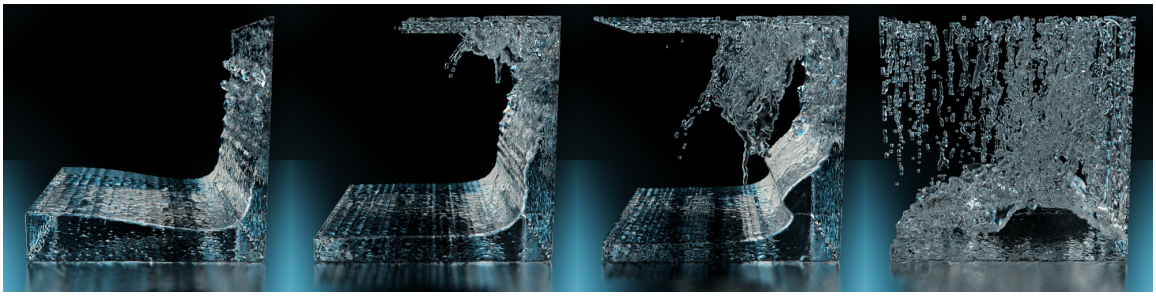
(e) $t = 120$ (f) $t = 160$ (g) $t = 200$ (h) $t = 249$

Figure 4.1: Breaking Dam Scenario Simulated by the Original CG Pressure Solve in 3D

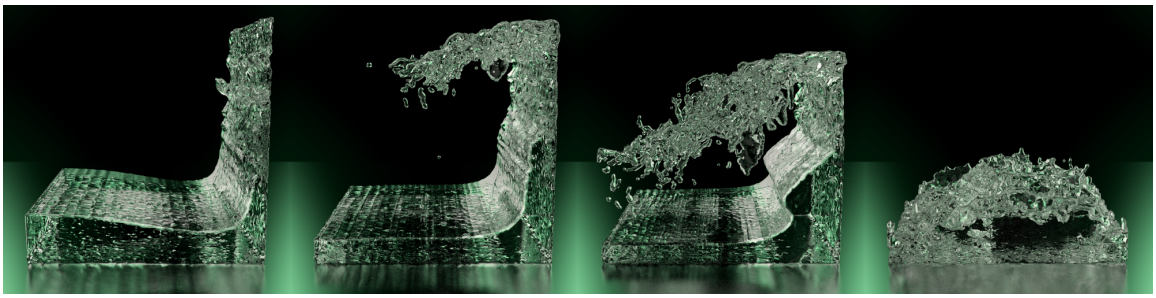
tions. Although the visual results vary for OrigPSSep, ADMMDivSep and IOPDivSep, all three methods produce visually plausible results and none seems more plausible than the other.
Against other predictions in [Bri08], it is therefore possible to achieve separating boundary conditions at fluid-solid boundaries with a simple CG solver when classifying the boundary cells into solid or air cells depending on their current role (solid wall or free-surface) before the pressure update.
Note that OrigPSSep advances faster than ADMMVelSep as demonstrated in Fig. 4.2 f), j) and g), k) while IOPDivSep can almost compete with OrigPSSep. At $t = 100$, the fluid is detaching from the ceiling. It seems that in OrigPSSep, the fluid movement describes a curve while simulations with ADMMVelSep and IOPDivSep show a linear fluid line. Furthermore, the fluid seems to behave less splashy for IOPDivSep, but all results look equally plausible. There is no obvious difference in mass loss during the simulation.
Simulating ADMM and IOP with sticky boundaries produces very similar results to OrigPS. The fluid is equally sticking to the walls and ceiling. However, it is impossible to determine which method is closer to OrigPS just from visually inspecting the simulation results. Some fluid sheets look more familiar when using ADMM, some fluid portion closer to IOP. Especially after many time steps, the differences in the resulting simulations are not easy to identify. The similarity of the methods is discussed in Section 4.3.

4.1 Convergence

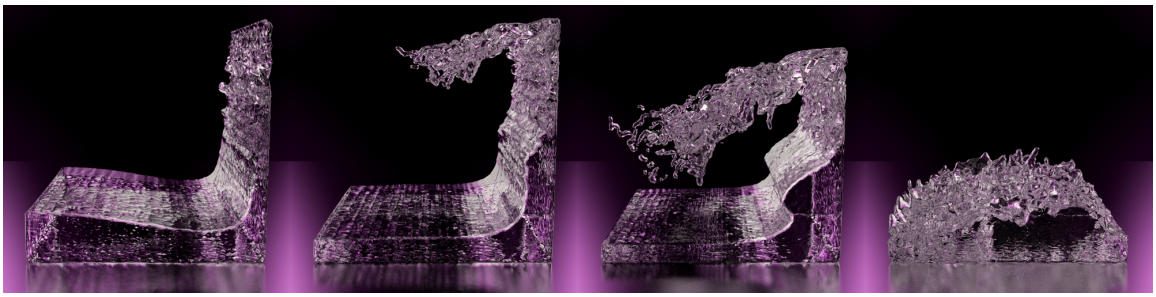
To compare the different ways of solving the pressure equations, the same stopping criterion should be used for accurate comparisons of the solutions and their performance. Unfortunately, this turns out to be very difficult. The common stopping criterion for ADMM is to measure the difference between both slack variables, the two velocity fields u and



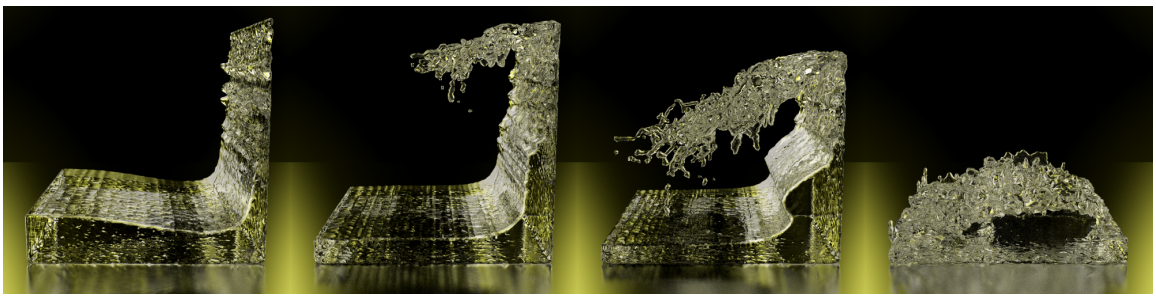
(a) OrigPS, $t = 66$ (b) OrigPS, $t = 100$ (c) OrigPS, $t = 120$ (d) OrigPS, $t = 160$



(e) OrigPSSep, $t = 66$ (f) OrigPSSep, $t = 100$ (g) OrigPSSep, $t = 120$ (h) OrigPSSep, $t = 160$



(i) ADMMVelSep, $t = 66$ (j) ADMMVelSep, $t = 100$ (k) ADMMVelSep, $t = 120$ (l) ADMMVelSep, $t = 160$



(m) IOPDivSep, $t = 66$ (n) IOPDivSep, $t = 100$ (o) IOPDivSep, $t = 120$ (p) IOPDivSep, $t = 160$

Figure 4.2: 3D Breaking Dam Simulations with Sticky and Separating Boundaries

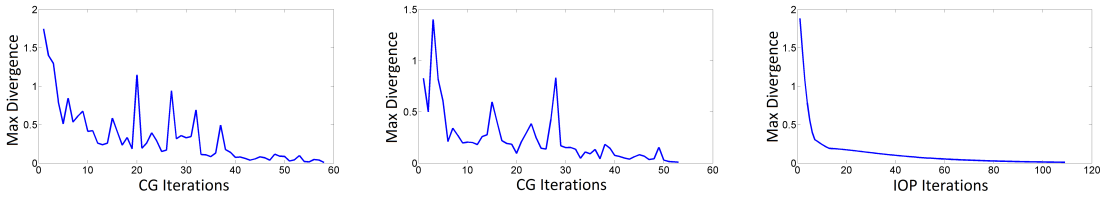
v. Minimizing the difference between both velocity fields leads to a solution where both problems are solved equally well. Since neither IOP nor the CG solver operate on two separate velocity fields, it is not possible to use the ADMM stopping criterion as a stopping criterion for all methods. When using the difference between the previous and the current velocity field $u_{current}$ and $u_{previous}$ in IOP, a different quantity is measured. While $r_{Avel} = u - v$ in ADMM measures how well both problems are solved simultaneously,

$r_{Ivel} = u_{current} - u_{previous}$ stands for the change within two IOP iterations which includes solving both problems in one single velocity field. Solving both problems in one velocity field might lead to reversing modification done by the pressure solve with the application of the boundary conditions. A much higher accuracy has to be chosen for IOP when using a stopping criterion based on the velocity change compared to running ADMM with r_{Avel} which is in turn hard to compare.
Another challenge occurs when inspecting the behavior of the residual of ADMM more deeply. The residual decreases in a periodic way with a period of approximately 20 time steps and an ever decreasing amplitude. It shows that the solutions of the two sub-problems alternate between approximating each other and drifting apart again but converge to the same solution in the end. An explanation therefore is the dual variable update which combines both solutions, taking one more into account than the other and overcorrecting this by weighting the other solution higher. It is thus important to stop the simulation at a point where both solutions are equally solved, i.e. when $r_{Avel} = u - v$ is at one of its minima.
Both the CG solver and the IOP method can use the divergence as stopping criterion. Due to the lack of another common stopping criterion, the divergence can also be used as a stopping criterion in ADMM but this comes with consequences. It is hence essential to examine where the divergence is introduced for each method.

Where does divergence occur?

1. **OrigPS**: When running a simulation with OrigPS, divergence is introduced mainly at the boundaries and dispersed over the whole field until it vanishes. Some single cells at the fluid surface show temporarily high divergence but go down uniformly during the CG iterations until the maximum value of divergence is below a specified threshold.
2. **OrigPSSep**: OrigPSSep shows the same divergence behavior as OrigPS but with additional divergence in the fluid cells next to solid cells which were classified as free-surface. This is expected since after reducing the divergence in the pressure solve, the pressure update is applied and creates new divergence by allowing the fluid to detach. When using the divergence as stopping criterion in OrigPSSep, the fluid cells next to solid cells must be disregarded in the divergence calculation since their divergence does not always go down. This divergence is fuels the motivation for applying ADMM and IOP.
3. **ADMMDivSep** and **IOPDivSep**: Most divergence in ADMM and IOP methods occurs at fluid cells next to solid cells. That is reasonable since the pressure solve eliminates the divergence and the boundary handling recreates divergence at fluid-solid faces by modifying the velocity there.

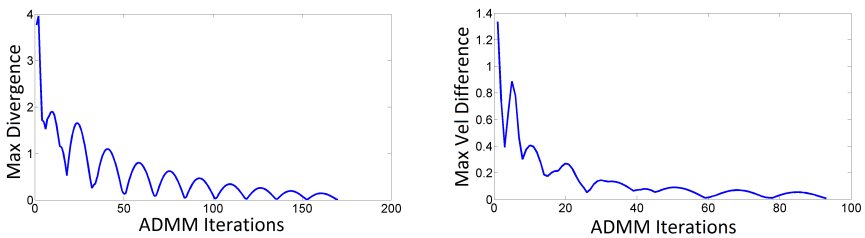
Divergence Residual for OrigPS, OrigPSSep and IOPDivSep The residual r_{div} in OrigPS and OrigPSSep goes down in a jagged way but reaches the desired threshold quite fast, see Fig. 4.3 (a) and (b). For IOP, the divergence goes down smoothly as shown in Fig. 4.3 (c). For each IOP iteration, the CG solver is called which minimizes the divergence up to the specified threshold. After that, the boundaries are handled which might introduce divergence again. The divergence decreases rapidly in the first steps and goes up again around the tenth step. From then on, it decreases straight as shown in Fig. 4.3 (c). Critical time steps like $t = 37$, see Fig. 4.1 (b), start with a higher divergence.



(a) OrigPS (b) OrigPSSep (c) IOPDivSep

Figure 4.3: Behavior of the Residual in time step $t = 150$

Divergence and Velocity Residual for ADMM The main problem with the divergence as stopping criterion for ADMM is, that it is a criterion considering how well the velocity field complies to the incompressibility condition and not taking into account how well the boundary conditions are met. This problem does not occur for IOP since both problems are solved in one single velocity field.
The divergence residual r_{div} as well as the usual residual r_{vel} develop in a periodic way. For some time steps, it takes a while to reach the periodic behavior and sometimes, the second amplitude is higher than the first. Especially critical time steps, e.g. $t = 37$ in Fig. 4.1 (b) where the fluid hits the right wall, show a high divergence in the beginning and the periodic behavior is reached not until the tenth iteration.
Stopping when the divergence is the lowest means to stop when the incompressibility problem is solved the best which does not imply that the boundary conditions are solved well at all. In fact, since both solutions approach each other and depart again, a minimum in divergence implies that the boundary conditions are not handled well. In Fig. 4.4 (a) ADMMDivSep with $r_{div} = div(u)$ is shown while in (b) ADMMVelSep with $r_{vel} = u - v$ is displayed. For better convenience, the choice of stopping criterion is appended to the method name. While the residual of ADMMVelSep has a minimum at $t = 78$ and at $t = 93$ amongst others, the divergence in ADMMDivSep peaks at $t = 75$ and $t = 92$ which proves that stopping by the means of r_{div} never results in a solution where both problems are solved well. The minima of the divergence are at $t = 67$, $t = 84$ and $t = 101$ amongst others. Therefore, the two residuals measure the same problem but a different quantity. While r_{Avel} measures how well both problems are solved, r_{Adiv} quantifies how well the incompressibility condition is met. As mentioned in Section 2.2, the variable q does converge to its optimal value but u and v not necessarily. The two residuals are phase shifted which makes it impossible to stop at minimum divergence and at the optimal solution for both problems. This is also indicated by simulations with ADMMDivSep which show slight artifacts



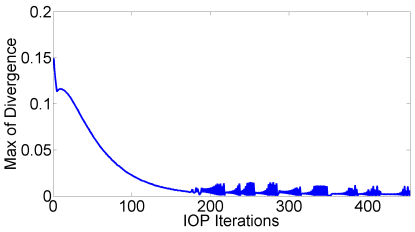
(a) ADMMDivSep, $\epsilon_{ADMM} = 10^{-2}$ (b) ADMMVelSep, $\epsilon_{Aabs} = \epsilon_{Arel} = 10^{-3}$

Figure 4.4: Behavior of the Residual in time step $t = 150$

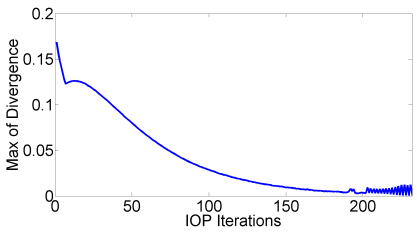
introduced due to slightly open boundaries while even lower accuracies for ADMMVelSep do not feature artifacts. It is recommended to use ADMM only with r_{Avel} to make sure both problems are solved well.
Note that r_{vel} is calculated before u is updated by the dual variable and r_{div} is measured after the update as explained in Section 3.2. The divergence plots in Fig. 4.3 and 4.4 are extracted from a simulation where OrigPSSep, ADMMVelSep, ADMMDivSep and IOPDivSep are all applied to the same input values. In each fifth step of an original simulation, the methods to compare are called on the same input values in order to provide a perfect comparison of several time steps. When all methods start in the same fluid state, the resulting fluid state can be compared accurately, see details in Section 4.3.

Choosing Thresholds for the Divergence Residual in CG, ADMMDiv and IOPDiv Using the divergence as stopping criterion leads to some restrictions regarding the choice of ϵ_{CG}, ϵ_{Adiv} and ϵ_{Idiv}. In each ADMM or IOP iteration, the CG pressure solve is called once. Both the internal CG and the outer ADMM or IOP loop terminate when the specified threshold for the divergence is undercut. If both the internal threshold ϵ_{CG} and the outer threshold ϵ_{Adiv} or ϵ_{Idiv} are equal, some difficulties occur. While the CG pressure solve reduces the divergence of u to a value below ϵ_{CG}, modifying u to comply with the boundary conditions introduces divergence which in turn means that the divergence might exceed ϵ_{CG} again. If ϵ_{Adiv} or ϵ_{Idiv} are equal to or even lower than ϵ_{CG}, the ADMM or IOP loop can hardly ever converge. The only possibility for it to converge is, when the CG pressure solver runs a minimum number of iterations and hence reduces the divergence eventually below ϵ_{Adiv} or ϵ_{Idiv}.
Therefore, the thresholds ϵ_{Adiv} and ϵ_{Idiv} should always be chosen less strict than ϵ_{CG}. It is better to have equally balanced iterations in the inner and the outer loop. Choosing ϵ_{Adiv} or ϵ_{Idiv} to be stricter than ϵ_{CG} leads to slowly decreasing divergence and a huge number or total CG iterations, especially for IOP. In Section 4.2, the performance is evaluated.

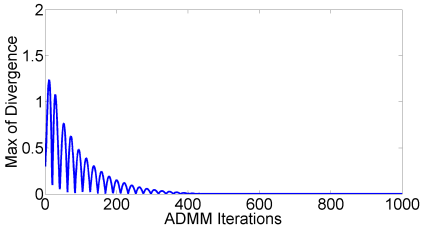
As shown in Fig. 4.5, when $\epsilon_{CG} = 10^{-3} > \epsilon_{Idiv}$, ϵ_{Adiv} the divergence approaches very slowly its threshold. While the divergence in ADMM develops in its usual periodic way



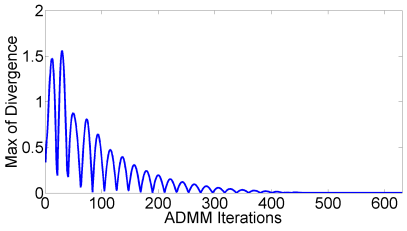
(a) IOPDivSep, $\epsilon_{Idiv} = 10^{-4}$, $t = 19$



(b) IOPDivSep, $\epsilon_{Idiv} = 5 \cdot 10^{-4}$, $t = 33$



(c) ADMMDivSep, $\epsilon_{Adiv} = 10^{-4}$, $t = 19$



(d) ADMMDivSep, $\epsilon_{Adiv} = 5 \cdot 10^{-4}$, $t = 33$

Figure 4.5: Behavior of the Residual with $\epsilon_{CG} = 10^{-3} > \epsilon_{Idiv}$, ϵ_{Adiv}

and only requires more iterations to undercut ϵ_{Adiv}, see Fig. 4.5 (c) and (d), the divergence in IOP behaves differently, see Fig. 4.5 (a) and (b). While it decreases smoothly until the two-hundredth iteration, it goes up and down it a jagged way until it eventually undercuts its threshold. This problem does not occur for ADMM with r_{Avel} since the stopping criteria in the inner and outer loop are not closely related.

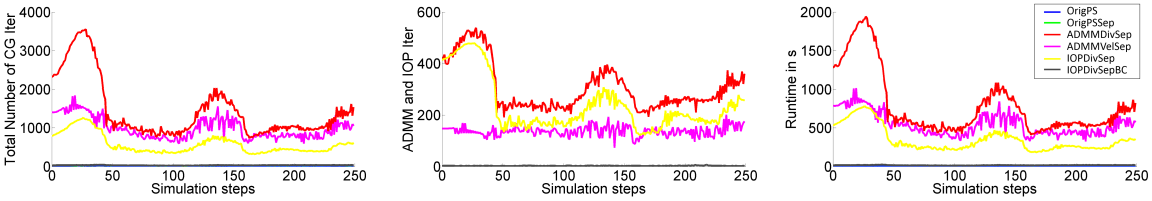
Choosing Thresholds for Velocity Residual in ADMMVel It turns out to be a good choice to set ϵ_{Aabs} and ϵ_{Arel} to a similar or even equal threshold. If one of the thresholds incorporates a much higher value, the other threshold is neglected.

4.2 Performance

In this section, the performance of OrigPS, OrigPSSep, ADMMVelSep, ADMMDivSep and IOPDivSep is evaluated by measuring the total amount of required CG, ADMM and IOP iterations, as well as the runtime per time step. In the first part, Section 4.2.1, the performance of the 3D breaking dam simulations from the beginning of this chapter is presented and evaluated. Next, Section 4.2.2, ADMMVelSep, ADMMDivSep and IOPDivSep are simulated in multiple, varying parameter settings. Thereby, the performance is examined depending on the choice of parameters, i.e. thresholds $\epsilon_{CG}, \epsilon_A, \epsilon_I$, and a practical parameter setting is determined.
The simulations with this very parameter setting are later on examined in Section 4.3, where all simulations are supplied with the same initial values for each time step. By doing so, the resulting output for each time step can be compared in a more convenient form than when the methods operate only at $t = 0$ on the same initial values. The methods are simulating a 2D breaking dam scenario.
The resulting performance of the settings chosen in Section 4.2.2 is presented in Section 4.2.3. As a last part, the performance is examined in relation to the chosen grid size in Section 4.2.4.

4.2.1 3D Breaking Dam Scenario

In the beginning of this chapter, a 3D simulation of the breaking dam scenario is run with OrigPS, OrigPSSep, ADMMVelSep and IOPDivSep, see Fig. 4.2. As a first evaluation aspect of the methods' performance, the averaged total number of CG iterations, the required number of ADMM and IOP iterations and the runtime measurements are visualized in Fig. 4.6. The averaged data is specified in Table 4.1. The performance of ADMMDivSep is examined as well in order to compare its performance to ADMMVelSep and IOPDivSep. The thresholds are set as follows: $\epsilon_{CG} = \epsilon_{Aabs} = \epsilon_{Arel} = 10^{-3}$ and $\epsilon_{Idiv} = \epsilon_{Adiv} = 10^{-2}$.
As expected, OrigPS and OrigPSSep need by far the least CG iterations, namely 20 CG iterations per time step and can hardly be recognized in Fig. 4.6. While the pressure solve is only applied once per time step, it is called several times from the ADMM and IOP loop when it comes to ADMMVelSep, ADMMDivSep and IOPDivSep simulations. ADMMDivSep requires the highest number of CG iterations, namely twice as many as IOPDivSep and factor 1.5 of ADMMVelSep, see Fig. 4.6 (a).
This confirms that using the divergence as residual in ADMM does not lead to optimal results. The performance plots for all three methods show a characteristic curve where the time steps $t = 0$ until $t = 50$, $t = 110$ until $t = 160$ and $t = 240$ until the end show higher resource consumptions compared to the others. Those time steps are therefore more difficult to solve. However, when it comes to the number of ADMM iterations in ADMMVelSep,



(a) Total Number of CG Iterations (b) ADMM and IOP Iterations (c) Runtimes

Figure 4.6: Iterations and Runtimes for the 3D Breaking Dam Simulation

Method	Color	Total CG Iter	ADMM / IOP Iter	Runtime in s
OrigPS	blue	20	-	6
OrigPSSep	green	20	-	10
ADMMVelSep	magenta	987	137	544
ADMMDivSep	red	1,446	305	790
IOPDivSep	yellow	574	236	348
IOPDivSepBC	gray	34	2.75	17

Table 4.1: Averaged Iterations and Runtimes per Time Step for 3D Breaking Dam

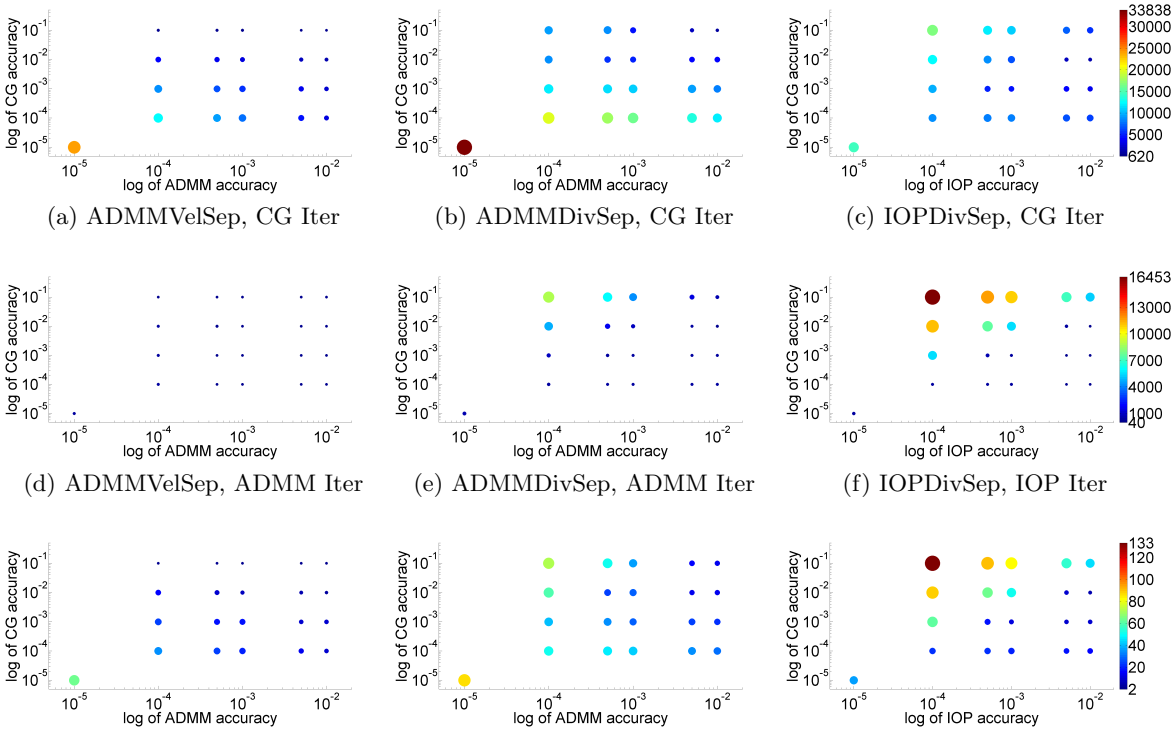
the number is very constant as shown in Fig. 4.6 (b). Only the inner CG loop takes more effort in solving those time steps.
The runtimes[1] are pointed out in Fig. 4.6 (c). ADMMVelSep requires 544 seconds per time step when simulating the 3D breaking dam scenario. This means that 250 time steps last for 37 hours only regarding the pressure solve. Even the best performing method as of IOPDivSep needs 50 times longer for a 3D breaking dam scene than OrigPS. Due to the poor performance, an acceleration method as hybrid between OrigPSSep and IOPDivSep is presented in Section 4.4. The accelerated method is called IOPDivSepBC and performs very well as shown in Table 4.1.
Furthermore, the performance of ADMMVelStick, ADMMDivStick and IOPDivStick is similar but leads to slightly higher resource consumption.

4.2.2 Varying Parameter Settings

To evaluate the performance of ADMMVelSep, ADMMDivSep and IOPDivSep in relation to the thresholds ϵ_{CG}, ϵ_{Aabs}, ϵ_{Arel}, ϵ_{Adiv} and ϵ_{Idiv}, all three methods are run 21 times with varying parameters.
The threshold ϵ_{CG} ranges from 10^{-1} to 10^{-5}, while ϵ_{Aabs}, ϵ_{Arel}, ϵ_{Adiv} and ϵ_{Idiv} take values between 10^{-2} and 10^{-5} as shown by the x- and y-axes in Fig. 4.7. Thereby, the highest CG, ADMM and IOP accuracy of 10^{-5} is only used once and serves as high-accuracy comparison. ϵ_{Aabs} and ϵ_{Arel} are always set to the same value. Since it is reasonable to assume that setting ϵ_{Aabs} and ϵ_{Arel} to the same value as ϵ_{Adiv} and ϵ_{Idiv} leads to comparable and slightly more accurate results, the same range is used for all three methods. The assumption about producing similar results is confirmed in Section 4.3. Due to the high resource consumption in 3D, these tests are run in 2D on a grid size of $96 \times 96 \times 1$.
In Fig. 4.6, the total amount of required CG iterations is displayed in the first row. The second row shows the total number of ADMM and IOP iterations while the runtime is

[1]The machine, on which all 3D simulations are run, is x64-based with Windows 8.1 Professional and an Intel(R) quad Core(MT) i7 CPU 920 @2.67GHz.

displayed in the last row [2]. The first aspect to notice when evaluating the total amount



(a) ADMMVelSep, CG Iter (b) ADMMDivSep, CG Iter (c) IOPDivSep, CG Iter

(d) ADMMVelSep, ADMM Iter (e) ADMMDivSep, ADMM Iter (f) IOPDivSep, IOP Iter

(g) ADMMVelSep, Runtime in s (h) ADMMDivSep, Runtime in s (i) IOPDivSep, Runtime in s

Figure 4.7: Performance of ADMM and IOP in Relation to the Thresholds ϵ

of CG iterations in the first row of Fig. 4.6 is that ADMMDivSep requires a lot of CG iterations, namely 33,838, for the high-accuracy simulation with $\epsilon_{CG} = \epsilon_{Adiv} = 10^{-5}$. In contrast to this, ADMMVelSep needs for $\epsilon_{CG} = \epsilon_{Aabs} = \epsilon_{Arel} = 10^{-5}$ only 24,500, whereas IOPDivSep requires the smallest number of CG iterations, that is 15,000. In contrast, for all lower combinations, e.g. $\epsilon_{CG} = \epsilon_{Aabs} = \epsilon_{Arel} = 10^{-3}$, ADMMVelSep needs equal or sometimes even less iterations compared to IOPDivSep.
Furthermore, the required CG iterations for settings above the diagonal from the lower left to the upper right are significantly higher than expected, especially for IOP as visualized in Fig. 4.6 (c). The simulation with $\epsilon_{CG} = 10^{-1}$ and $\epsilon_{Idiv} = 10^{-4}$ requires 16,743 CG iterations while the high-accuracy comparison simulation converges with 14,842 CG iterations. This demonstrates that IOP converges poorly when choosing a higher accuracy for the outer loop than for the inner loop. Although ADMMDivSep usually needs almost double as many CG iterations than IOPDivSep with the same thresholds, ADMMDivSep needs only 9.637 CG iterations for $\epsilon_{CG} = 10^{-1}$ and $\epsilon_{Adiv} = 10^{-4}$. That lets conclude that ADMMDivSep converges slowly but more robustly than IOP.
The IOP iterations in Fig. 4.7 (f) match the observations based on plot Fig. 4.7 (c). IOP requires 16,453 outer iterations for $\epsilon_{CG} = 10^{-1}$ and $\epsilon_{Idiv} = 10^{-4}$ which is almost equal to the number of CG iterations. That confirms the aforementioned assumption that in the pressure solve, only the minimum number of iterations are conducted because the divergence of u is already below ϵ_{CG}. A remarkable fact hereby is, that the runtime is increased depending on both the number of CG iterations and IOP iterations but the number of IOP

[2] On a x64-based machine with Windows 8.1 Professional, Intel(R) Xeon(R) 6 Core CPU E5-1650 v2 @3.50GHz.

iterations weight much more for IOP as shown in Fig. 4.7 (i).
ADMMDivSep requires a high number of ADMM iterations as well when choosing a higher ADMM accuracy than for the pressure solve as shown in Fig. 4.7 (e). For $\epsilon_{CG} = 10^{-1}$ and $\epsilon_{Adiv} = 10^{-4}$, 8,911 ADMM iterations are required which is only half as many IOP iterations for the same setting. The increased number of ADMM iterations afflicts the runtime as well as shown in Fig. 4.7 (h). But compared to IOP, ADMM handles the threshold settings visualized above the diagonal a lot better. Hence, it is confirmed that the outer accuracy should always be lower or equal than the inner accuracy. Choosing the same inner as outer accuracy leads to few peaks in the number of ADMM and IOP iterations but does not strikes down in the runtime.
The issue of selecting similar thresholds for the inner CG and outer ADMM loop does not occur for ADMMVelSep as shown in Fig. 4.7 (a). If the pressure solve does not modify u much, both velocity fields u and v approach each other faster and hence, convergence is reached soon. Independent of the value of ϵ_{CG}, only related to ϵ_{Aabs} and ϵ_{Arel}, the number of ADMM iterations is stable between 40 and 200 and goes up to 300 for the high-accuracy comparison solution as shown in Fig. 4.7 (d). ADMMVelSep needs roughly a 30% more CG iterations than IOP for ADMM accuracies higher or equal to 10^{-4}, but less iterations for e.g. $\epsilon_{Aabs} = \epsilon_{Arel} = 10^{-2}$. For the high-accuracy simulation, 24,473 CG iterations are required with ADMMVelSep which is much more than compared to IOPDivSep. As mentioned in Section 2.2, ADMM converges best for moderate accuracies.
The high-accuracy simulation needs 87 seconds per time step which makes 6 hours for the pressure solve for ADMMDivSep when simulating the breaking dam scenario with $t_{max} = 250$. ADMMVelSep requires 65 seconds per time step and IOPDivSep 19 seconds which leads to a total amount of required time of 4.5h, respectively 1.3h.

The most robust variant in terms of convergence is ADMMVelSep. When choosing a moderate accuracy, the performance is quite good.
While Fig. 4.7 visualizes the number of CG iterations, the number of ADMM and IOP iterations and runtime per time step, Table 4.2 lists the exact number of CG iterations. Since some simulations feature slight artifacts due to too weak accuracy, the corresponding table entries are colored to exclude this parameter setting from being considered as a practical solution. Magenta colored cells signalize that the corresponding ADMMVelSep simula-

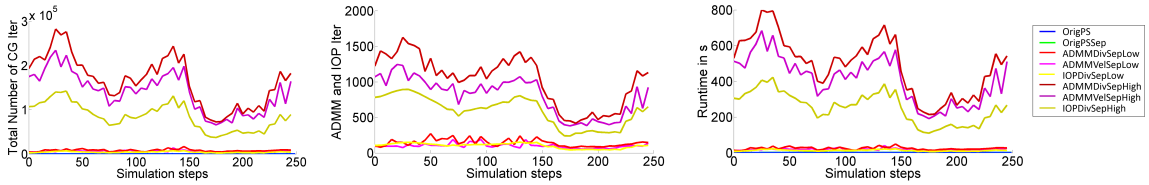
ϵ_{CG} \ $\epsilon_{A/I}$	10^{-5}	10^{-4}	$5 \cdot 10^{-4}$	10^{-3}	$5 \cdot 10^{-3}$	10^{-2}
10^{-5}	i_{Avel}=24,473 i_{Adiv}=33,838 i_{Idiv}=14,842					
10^{-4}		i_{Avel}=12,915 i_{Adiv}=20,170 i_{Idiv}=9,053	i_{Avel}=9,387 i_{Adiv}=18,204 i_{Idiv}=8,731	i_{Avel}=7,884 i_{Adiv}=16,374 i_{Idiv}=8,389	i_{Avel}=4,752 i_{Adiv}=13,822 i_{Idiv}=6,989	i_{Avel}=3,399 i_{Adiv}=12,199 i_{Idiv}=6,199
10^{-3}		i_{Avel}=9,176 i_{Adiv}=12,449 i_{Idiv}=10,107	i_{Avel}=6,704 i_{Adiv}=11,809 i_{Idiv}=5,162	i_{Avel}=5,650 i_{Adiv}=10,950 i_{Idiv}=4,680	i_{Avel}=3,568 i_{Adiv}=9,478 i_{Idiv}=4,266	i_{Avel}=2,611 i_{Adiv}=8,419 i_{Idiv}=3,882
10^{-2}		i_{Avel}=4,568 i_{Adiv}=9,309 i_{Idiv}=12,872	i_{Avel}=3,927 i_{Adiv}=5,838 i_{Idiv}=9,189	i_{Avel}=3,380 i_{Adiv}=5,206 i_{Idiv}=7,192	i_{Avel}=2,391 i_{Adiv}=4,458 i_{Idiv}=1,993	i_{Avel}=1,693 i_{Adiv}=4,383 i_{Idiv}=1,586
10^{-1}		i_{Avel}=837 i_{Adiv}=9,637 i_{Idiv}=16,743	i_{Avel}=851 i_{Adiv}=6,700 i_{Idiv}=12,138	i_{Avel}=844 i_{Adiv}=4,789 i_{Idiv}=11,167	i_{Avel}=712 i_{Adiv}=2,128 i_{Idiv}=7,327	i_{Avel}=620 i_{Adiv}=1,295 i_{Idiv}=5,642

Table 4.2: Total Number of CG Iterations for Different Parameter Settings

tion features some artifacts while a gray cell color stands for artifacts in ADMMDivSep. IOPDivSep never features visual artifacts but many parameter settings cannot be considered due to the imbalanced number of iterations in the inner and outer loop. Despite the slight artifacts, all simulations exhibit a similar and visually plausible fluid motion.
By evaluating the performance plots in Fig. 4.6 and Table 4.2, a practical setting can be determine. This practical setting is chosen as low as possible. For ADMMDivSep as well as for IOPDivSep, $\epsilon_{CG} = 10^{-3}$, $\epsilon_{Adiv} = 10^{-2}$ and $\epsilon_{Idiv} = 10^{-2}$ is chosen as lowest setting in terms of accuracy where neither IOP shows poor convergence nor does ADMM show artifacts. Choosing a parameter setting for each method independently might lead to different thresholds.

4.2.3 Identical Input Values in Each Time Step

In this section, the performance of 2D simulations of the breaking dam scenario is analyzed. In practice, parameter ϵ_{CG} is usually set to 10^{-5}. However, since ADMM works the best if both ϵ_{CG} and ϵ_A are set to a similar accuracy and ADMM works best with moderate accuracy, the threshold ϵ_{CG} must be increased to 10^{-3} to lower the resource consumption and therefore enable the evaluation of many different scenarios.
For example, in Section 4.3 is examined if the method's solutions approach each other for very high accuracy, i.e. if there exists an agreement solution. To examine this, an original simulation is run and in every fifth time step, OrigPSSep, ADMMVelSep, ADMMDivSep and IOPDivSep are run on the same input variables generated by OrigPS. It is ensured that all methods start with the same initial state and therefore, their output state can be compared very well. ADMMVelSep, ADMMDivSep and IOPDivSep are run once with very high accuracy, $\epsilon_{CG} = 10^{-12}$ and $\epsilon_{Aabs} = \epsilon_{Arel} = \epsilon_{Adiv} = \epsilon_{Idiv} = 10^{-9}$, and with low, practical, accuracy $\epsilon_{CG} = 10^{-3}$, $\epsilon_{Aabs} = \epsilon_{Arel} = 10^{-3}$ and $\epsilon_{Adiv} = \epsilon_{Idiv} = 10^{-2}$. The



(a) CG Iterations (b) ADMM and IOP Iterations (c) Runtimes in s

Figure 4.8: Performance of 2D Breaking Dam with Same Input Values

Method	Color	Total CG Iter	ADMM / IOP Iter	Runtime in s
OrigPSHigh	blue	321	-	1
OrigPSSepHigh	green	290	-	1
ADMMVelSepHigh	dark magenta	140,399	808	412
ADMMVelSepLow	magenta	6,196	99	18
ADMMDivSepHigh	dark red	165,559	1,027	491
ADMMDivSepLow	red	8209	142	25
IOPDivSepHigh	dark yellow	84,929	592	254
IOPDivSepLow	yellow	4,280	102	13

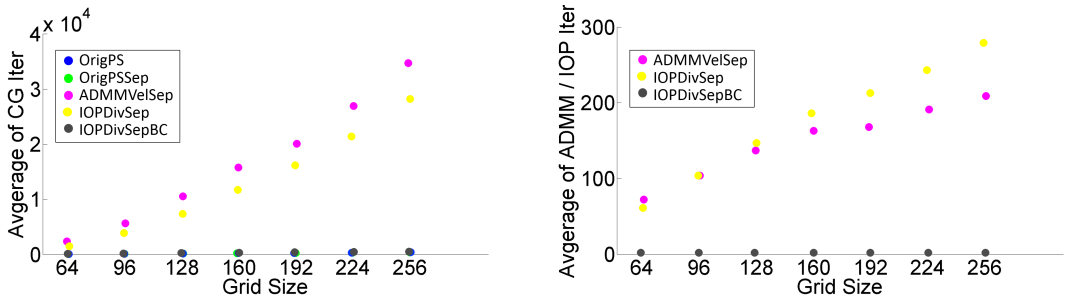
Table 4.3: Averaged Iterations and Runtimes per Time Step for 3D Breaking Dam

required number of CG iterations, ADMM and IOP iterations are displayed in Fig. 4.8 (a)

and (b). The corresponding runtimes in seconds per time step[3] is shown in Fig. 4.8 (c). The performance of OrigPS and OrigPSSep is obviously better than the performance of ADMM and IOP since the pressure solve is only called once and does not operate in a loop. The maximum averaged number of iterations per time step is 320 for OrigPS with $\epsilon_{abs} = 10^{-12}$. The performance of OrigPSSep is slightly better, which can be inferred from Table 4.3.
ADMMDivSep is again the most expensive method with twice or a quarter as many CG iterations as IOPDivSep or ADMMVelSep. From the overall perspective, all plots show the same characteristic curve as before, consuming most resources for difficult time steps.
Note that ADMMVelSepLow needs less ADMM iterations than IOPDivSepLow needs IOP iterations but consumes a quarter more CG iterations. These plots confirm that IOPDivSep is the most efficient method.

4.2.4 Varying Grid Sizes

In order to analyze the behavior of the most important methods, i.e. OrigPS, OrigPSSep, ADMMVelSep, IOPDivSep and IOPDivSepBC, the grid size is varied in this section as shown in Fig. 4.9. It Both the total amount of CG iterations for ADMMVelSep and IOPDivSep increase in an exponential manner, although IOP requires always less CG iterations than ADMM. When it comes to ADMM and IOP iterations, the iterations increase in a rather linear way and the ADMM iterations are lower than IOP iterations for grids larger than 96×96. Since the number of CG iterations does increase almost exponentially, the 3D simulations in this thesis are restricted to the grid size $96 \times 96 \times 1$. Increasing the grid size to $128 \times 128 \times 128$ leads to four times higher runtime requirements, which is not feasible in the scope of this thesis.
The number of CG iterations required for OrigPS increase from 76 to 410 while OrigPSSep uses 72 to 362 iterations. The thresholds are as usual as follows: $\epsilon_{CG} = \epsilon_{Aabs} = \epsilon_{Arel} = 10^{-3}$ and $\epsilon_{Idiv} = 10^{-2}$.



(a) CG Iterations

(b) ADMM and IOP Iterations

Figure 4.9: Number of Iterations over Grid Size

4.3 Exactness and Similarity of Solutions

Since there is no stopping criterion equally suitable for all three methods, OrigPS, ADMM and IOP can hardly be compared in terms of performance and exactness of the resulting

[3] On a x64-based machine with Windows 8.1 Professional, Intel(R) Xeon(R) 6 Core CPU E5-1650 v2 @3.50GHz.

solution. However, when an agreement solution, to which all solutions converge, is found, two methods with different stopping criteria and accuracy can indeed be reasonably compared.
The most important methods to be compared are OrigPSSep, ADMMVelSep, ADMMDivSep and IOPDivSep but for evaluating how close ADMM and IOP can get to OrigPS itself, ADMMVelStick, ADMMDivStick and IOPDivStick are evaluated as well. Therefore, both sticky and separating boundary conditions are evaluated.
The first point of interest is, if all three methods are solving the same problem when running very long, i.e. if there exists an agreement solution to which all methods converge if the accuracies are set high enough, e.g. $\epsilon_{CG} = 10^{-12}$, $\epsilon_{Adiv} = \epsilon_{Idiv} = 10^{-9}$ and $\epsilon_{Aabs} = \epsilon_{Arel} = 10^{-9}$. The next task is then to evaluate how close the methods with practical accuracy, i.e. $\epsilon_{CG} = 10^{-3}$, $\epsilon_{Adiv} = \epsilon_{Idiv} = 10^{-2}$ and $\epsilon_{Aabs} = \epsilon_{Arel} = 10^{-3}$ approach this agreement solution to figure out the most accurate solution. When two methods with specified accuracy reach a similar solution, their performance can be reasonably compared.

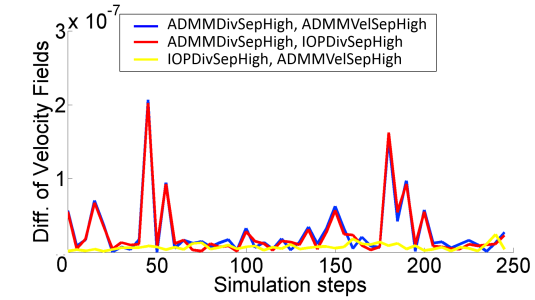
OrigPS, ADMMVelStick, ADMMDivStick and IOPDivStick To evaluate the similarity between the original pressure solve OrigPS and ADMM and IOP, the methods OrigPS, ADMMVelStick, ADMMDivStick and IOPDivStick are run with the high accuracies ($\epsilon_{CG} = 10^{-12}$, $\epsilon_{Adiv} = \epsilon_{Idiv} = 10^{-9}$ and $\epsilon_{Aabs} = \epsilon_{Arel} = 10^{-9}$). The results are shown in Fig. 4.10 (b). The differences in the resulting velocities are in the range of 10^{-7} to 10^{-9} which is a very good result. It means that all methods are solving the same problem. In- or decreasing the accuracies leads to more or less similar results. The differences are in order of $max(\epsilon_{CG}, \epsilon_{Adiv} or \epsilon_{Idiv} or max(\epsilon Aabs, \epsilon_{Arel})$. Therefore, the solutions can get arbitrarily similar by varying the accuracies - all three methods converge to an agreement solution.
It can be observed that ADMMDivStick differs the most from OrigPS and ADMMVelStick is the closest.

OrigPSSep, ADMMDivSep, ADMMVelSep, IOPDivSep It is of interest to know if OrigPSSep does also converge to the agreement solution even though it introduces divergence and does not iteratively reduce the divergence as ADMMDivSep and IOPDivSep. As shown in Fig. 4.10 (d), OrigPSSep does not converge to the same solution as ADMMDivSep, ADMMVelSep and IOPDivSep when all are running with a very high accuracy. The differences in the resulting velocity fields are in the range of 10^0 and do not decrease if lower accuracies are chosen for ADMMDivSep, ADMMVelSep and IOPDivSep as displayed in Fig. 4.10 (f). While in (d), all three methods differ equally far from OrigPSSep, ADMMDivSep differs a bit more when run with a low accuracy ind (f). Nevertheless, OrigPSSep delivers a completely different solution compared to simulations with ADMM and IOP. The differences especially rise after the first hit onto the wall aroung $t = 37$, before that, no cell was classified as detaching.

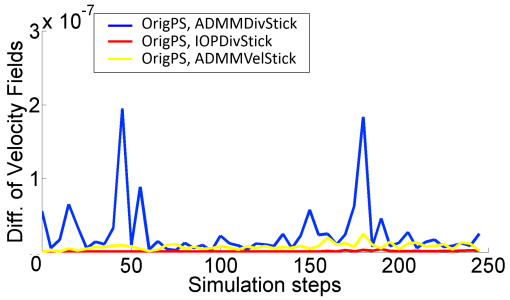
ADMMDivSep, ADMMVelSep, IOPDivSep In this paragraph, an evaluation regarding the convergence to an agreement solution of ADMMDivSep, ADMMVelSep and IOPDivSep is conducted. Especially the difference between using the velocity and the divergence as stopping criterion in ADMM must be examined. Since ADMMVelSep and IOPVelSep provide the closest solutions as shown in Fig. 4.10 (a), the divergence was justifiably ruled out as stopping criterion in 4.1 for ADMM. The differences of all solutions reside in the range of 10^{-8} to 10^{-9}, IOPDivSep and ADMMVelSep differ in the range of 10^{-9}, which is exactly the threshold ϵ_{Idiv} and $\epsilon_{Arel} = \epsilon_{Aabs}$. ADMMDivSep converges slower to the agreement solution but it does converge since the difference is getting smaller with higher

accuracy. In Fig. 4.10 (c), all three methods are run with lower, practical accuracy. The difference ranges in 10^{-2} and 10^{-3} which is exactly $max(\epsilon_{Idiv}, \epsilon_{Adiv}, \epsilon_{Aabs}, \epsilon_{Arel})$. IOPDivSep and ADMMVelSep are again the closest solutions. Since ADMMVelStick is also the closest to OrigPS, ADMMVelSep is now selected as agreement solution although IOPDivSep would be perfectly suitable as well. In Fig. 4.10 (e), the practical ADMM and IOP simulations are compared to the agreement solution. As expected, ADMMDivSep differs the most from the agreement solution. ADMMVelSep is the most accurate solution except for one time step around $t = 70$. IOPDivSep provides the most accurate result for this time step.

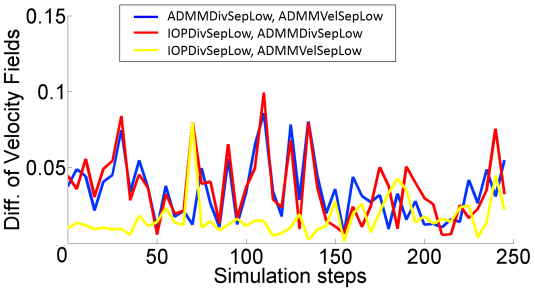
To investigate the source of the velocity differences between the practical execution of ADMM and IOP, some difference fields of the resulting velocity fields are visualized in Fig. 4.11, enlarged by the factor of 200 to make the differences clearly visible. The time steps $t = 70$, $t = 125$ and $t = 195$ are selected to be investigated deeper, since most difference plots show high differences around $t = 195$, the deviation of ADMMVelSepLow to the agreement solution in Fig. 4.10 (e) occurs at $t = 70$ and because $t = 125$ shows



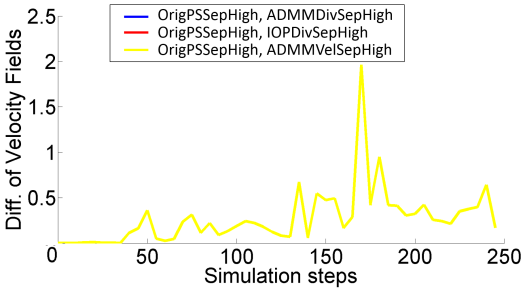
(a) Agreement Solution



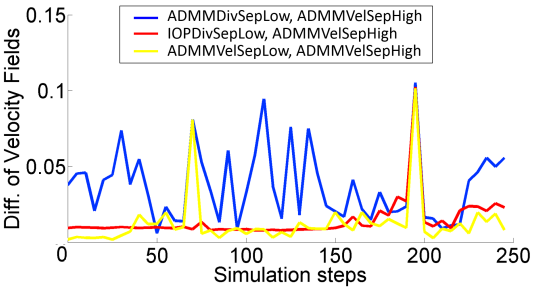
(b) OrigPS



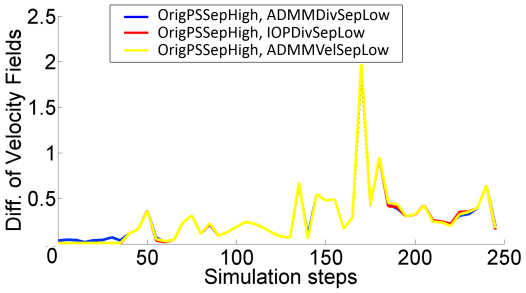
(c) Similarity of Lower Accuracies



(d) OrigPSSep with Others High



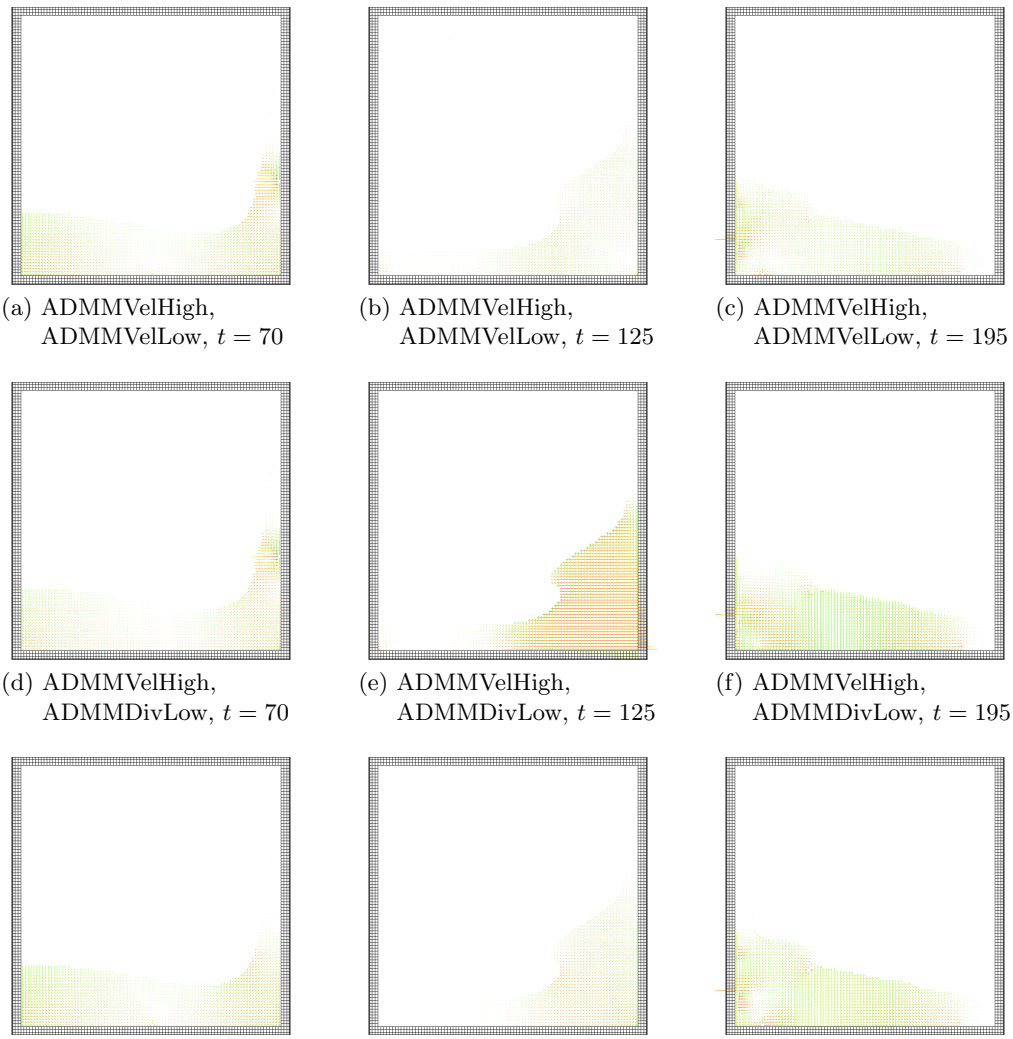
(e) Similarity of Lower Accuracies to Agreement Solution



(f) OrigPSSep with Others Low

Figure 4.10: Similarities

high differences between ADMMDivSep and the agreement solution. Velocities in the x-directions are painted red, while velocities in the y-directions have a green color. Velocities where both directions are equally represented have therefore a yellow color.
In the first time step, $t = 70$, the fluid has crawled half up the right wall and may start to detach. The differences in the velocity fields in Fig. 4.11 (a) and (d) are very similar. The fluid cell with index (35, 92) (grid size $96 \times 96 \times 1$) shows a large negative velocity into the x-direction which explains the peak in Fig. 4.10 (c) and (e). This shows that lower accuracy ADMM simulations classify the neighboring solid cell as detaching while the agreement solution leaves it as solid wall with zero velocity. IOPDivSep seems to classify it correctly even with low accuracy. The difference between ADMMDivSep and the agreement solution is particularly high in time step $t = 125$ where the fluid is just forming a wave towards the left wall. The difference is equally distributed over the right part of the velocity field. Note that this occurs in the main fluid body and not in single drops detaching from the wall. The comparison plots usually look like this frame, the other two time steps represent exceptional time steps which can also be seen in Fig. 4.10. When the main fluid part has



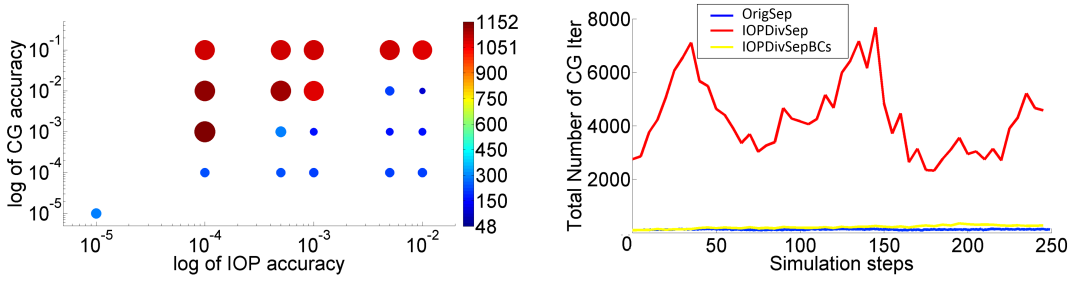
(a) ADMMVelHigh, ADMMVelLow, $t = 70$
(b) ADMMVelHigh, ADMMVelLow, $t = 125$
(c) ADMMVelHigh, ADMMVelLow, $t = 195$
(d) ADMMVelHigh, ADMMDivLow, $t = 70$
(e) ADMMVelHigh, ADMMDivLow, $t = 125$
(f) ADMMVelHigh, ADMMDivLow, $t = 195$
(g) ADMMVelHigh, IOPDivLow, $t = 70$
(h) ADMMVelHigh, IOPDivLow, $t = 125$
(i) ADMMVelHigh, IOPDivLow, $t = 195$

Figure 4.11: Differences of Resulting Velocity Fields

splashed against the left wall at $t = 195$, another high difference occurs, this time in all three comparisons. This time step seems to be quite difficult since all difference plots in Fig. 4.10 show higher peaks shortly before $t = 200$. The main difference occurs in the 16^{th} row from the bottom in the x-direction of the velocity. After fluid has splashed against a solid wall, the methods seem to have difficulties in resolving the pressure in an accurate way. The difference for ADMMDivLow in Fig. 4.11 (e) shows additionally higher differences in the y-direction around the center of the fluid.
Overall, the fact that ADMMDivSep differs from ADMMVelSep and IOPDivSep confirms the observation of the phase shifted residual when using the divergence as stopping criterion. As mentioned before, ADMMDivSep sometimes shows artifacts due to the lack of boundary conditions compliance.

4.4 Acceleration of ADMM and IOP

To accelerate the performance of ADMM and IOP, the approach to pass information about solid wall boundary conditions from the ADMM and IOP loop to the pressure solve was made. However, since the solid wall boundary criteria are already met, the second proximal operator *ensureBCs* will not have any effect in the velocity field which redundantizes the second velocity field v in ADMM. Passing solid wall boundary conditions to the pressure solve means basically to apply OrigPSSep several times and to reduce therefore the introduced divergence until the specified threshold is met. The only suitable stopping criterion in this case is the divergence. ADMM and IOP obviously deliver the same solution. Since only one velocity field is needed, this combination of IOPDivSep and OrigPSSep is now called IOPDivSepBC where BC stands for boundary conditions.
Nevertheless, the choice of thresholds for the outer and the inner loop must be respected as well. Higher IOP accuracy compared to the inner CG accuracy leads to a high number of total CG iterations as visualized in Fig. 4.12 (a) which strikes down on the runtime identically. Since this inefficient behavior of IOP is known from Section 4.1 and 4.2.2, the maximum number of IOP iterations is set to 1000, otherwise, the number of iterations above the diagonal would be even higher. In Table 4.4, the corresponding data to the number of iterations in Fig. 4.12 (a) is shown.



(a) Different Accuracies for IOPDivSepBC

(b) $\epsilon_{CG} = 10^{-3}$, $\epsilon_{Idiv} = 10^{-2}$

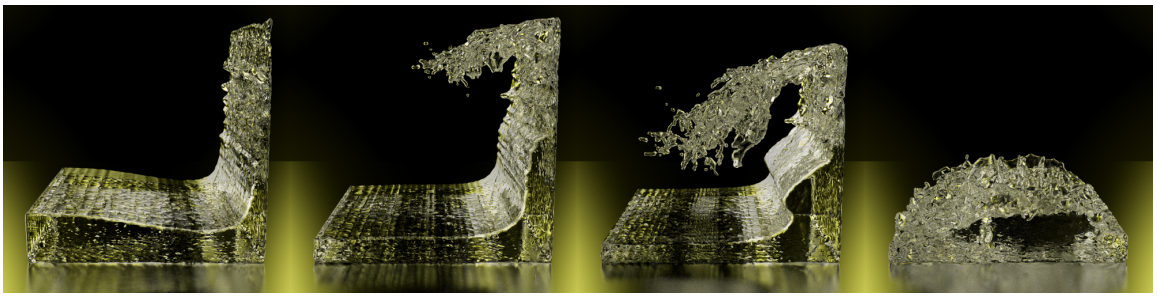
Figure 4.12: Total Number of CG iterations per Time Step

While IOPDivSep requires 1400 CG iterations to achieve the accuracy of $\epsilon_{CG} = 10^{-5}$ and $\epsilon_{Idiv} = 10^{-5}$ and approximately 3900 CG iterations for $\epsilon_{CG} = 10^{-3}$ and $\epsilon_{Idiv} = 10^{-2}$, IOPDivSepBC needs only 305 or 192 respectively which is a huge difference and makes the simulation very feasible. OrigPSSep introduces divergence which is perfectly eliminated in

ϵ_{CG} \ $\epsilon_{A/I}$	10^{-5}	10^{-4}	$5 \cdot 10^{-4}$	10^{-3}	$5 \cdot 10^{-3}$	10^{-2}
10^{-5}	$i_{IBC} = 305$					
10^{-4}		$i_{IBC} = 253$	$i_{IBC} = 251$	$i_{IBC} = 248$	$i_{IBC} = 245$	$i_{IBC} = 239$
10^{-3}		$i_{IBC} = 1152$	$i_{IBC} = 218$	$i_{IBC} = 182$	$i_{IBC} = 183$	$i_{IBC} = 192$
10^{-2}		$i_{IBC} = 1127$	$i_{IBC} = 1112$	$i_{IBC} = 1054$	$i_{IBC} = 241$	$i_{IBC} = 119$
10^{-1}		$i_{IBC} = 1067$	$i_{IBC} = 1068$	$i_{IBC} = 1067$	$i_{IBC} = 1061$	$i_{IBC} = 1040$

Table 4.4: Total Number of CG Iterations for Different Parameter Settings

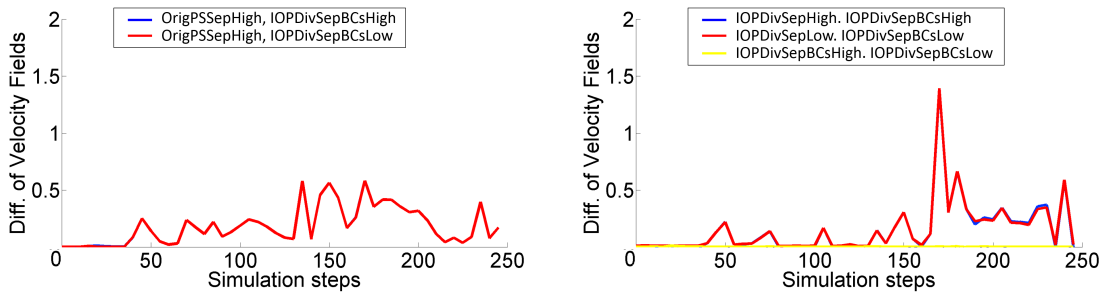
IOPDivSepBC. The visual results of IOPDivSepBC are shown in Fig. 4.13 by the means of the 3D breaking dam simulation as in the beginning of this chapter. The generated simulation is visually very plausible and no differences can be detected compared to Fig. 4.2. To



(a) IOPDivSep, $t = 66$ (b) IOPDivSep, $t = 100$ (c) IOPDivSep, $t = 120$ (d) IOPDivSep, $t = 160$

Figure 4.13: 3D Breaking Dam Simulations with Sticky and Separating Boundaries with Boundary Information in CG

evaluate these results accurately, OrigPSSep, IOPDivSep and IOPDivSepBC are applied to the same input values as performed before in Section 4.3. The performance comparison of all three methods is demonstrated in Fig. 4.12 (b). While OrigPSSep needs 122 total CG iterations as shown in Table 4.3, IOPDivSepBC needs 213 which is almost twice as much. Averaged over all time steps, 2 IOP iterations are needed which coincides with the number of the CG iterations. Compared to IOPDivSep which needs 4290 CG iterations, the performance is drastically improved.
Since all methods produce visually plausible results with no identifiable differences, the similarity of OrigPSSep, IOPDivSepBC and IOPDivSep is evaluated, see Fig. 4.14. OrigPSSep



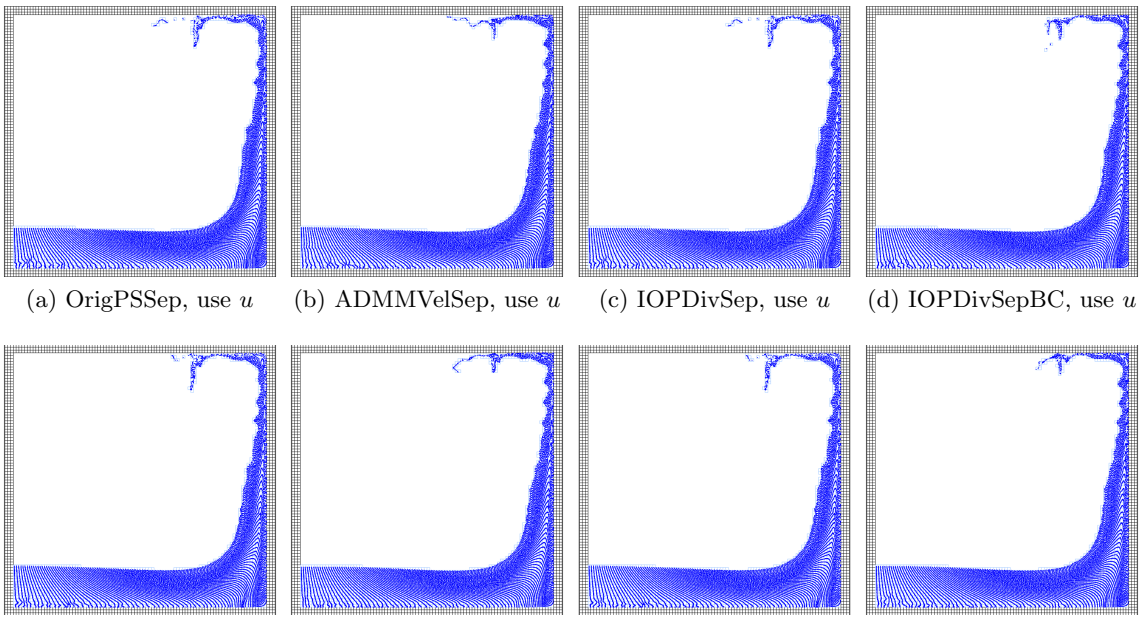
(a) OrigPSSep vs. IOPDivSepBC (b) IOPDIvSep vs. IOPDivSepBC

Figure 4.14: Difference in the Velocity Fields (velocity in cell units)

and IOPDivSepBC exhibit a quite large difference as shown in Fig. 4.14 (a). This might result from removing the divergence at fluid-solid faces. Since the difference is almost equal for low and high accuracies for IOPDivSepBC, the eliminated divergence is a reasonable explanation. The difference between IOPDivSep and IOPDivSepBC is even bigger as shown in Fig. 4.14 (b). Independently of the chosen thresholds, the difference is highest around $t = 175$ and develops in the same pattern. However, it is very interesting to see that IOPDivSepBC produces very similar results. The differences lie in the range of 10^{-3} to 10^{-4} which is exactly the lowest accuracy of both simulations.
Including information about boundary conditions in the pressure solve leads to efficient methods for simulating fluids with separating boundaries without introducing divergence. But note that including information about the boundary conditions makes the separation of the two sub-problems impossible and can therefore not be solved by applying ADMM.

4.5 Alternative Cell Classification Based on Pressure

As aforementioned in Section 3.2, the classification of solid cells can be conducted by means of the velocity or pressure. In this thesis, solid cells are classified as free-surface in ADMM and IOP when the normal velocity component at fluid-solid faces is positive after the pressure solve. The classification is based on the velocity to allow for pressure solvers that only return velocity values and no pressure values. In OrigPSSep, the pressure equations are first solved and a hypothetical pressure update is performed where all solid cells are set to air cells. Those solid cells that feature positive normal velocity components at fluid-solid faces, are then classified as free-surface.
Another possibility is to classify solid cells as free-surface when the pressure in adjacent fluid cells is negative or zero. In Fig. 4.15, the resulting simulations of OrigPSSep, ADMMVelSep, IOPDivSep and IOPDivSepBC are shown and compared to the original classification based on velocities. It seems that a classification based on the velocity leads



(a) OrigPSSep, use u (b) ADMMVelSep, use u (c) IOPDivSep, use u (d) IOPDivSepBC, use u

(e) OrigPSSep, use p (f) ADMMVelSep, use p (g) IOPDivSep, use p (h) IOPDivSepBC, use p

Figure 4.15: 2D Breaking Dam Scene with Varying Solid Cell Classification, $t = 85$

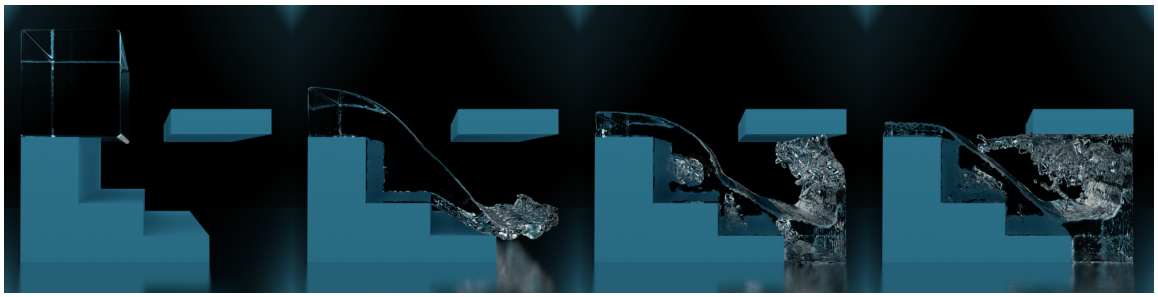
to slightly better detaching fluid as shown in the first row in Fig. 4.15 (a)-(d). A classification based on pressure does produce very similar results, see Fig. 4.15 (e)-(h).

4.6 Obstacles and Robustness

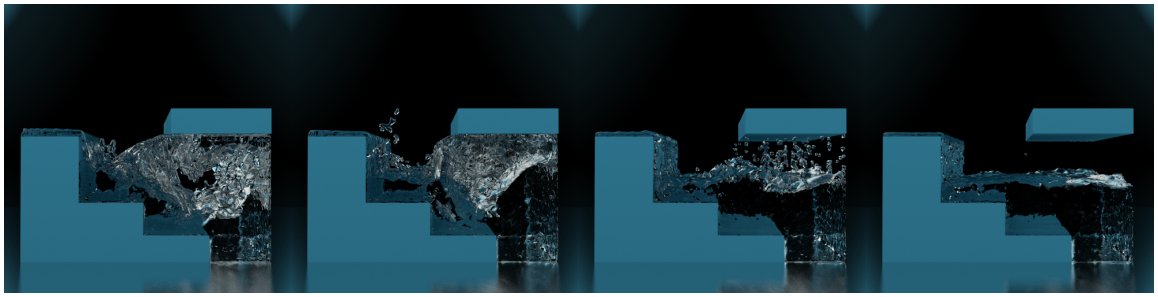
To evaluate the robustness of OrigPSSep, ADMMVelSep, IOPDivSep and IOPDivSepBC, additional scenarios are simulated and evaluated in this section. Thereby, the choice of thresholds is validated.
The first scenario in Section 4.6.1 is called stair since three cuboids are placed in the simulation domain which function as stair steps. To test the fluid's detaching behavior, a floating plane is placed in the middle of the domain which the fluid hits from below. The second scenario in Section 4.6.2 describes two fluid blocks flowing down from two levitating planes in the upper part of the domain. A sphere is placed in the middle so that the fluid is flowing around it. The floor is peppered with small cubes. The fluid is then flowing through the aisles between the cubes. As a last scenario, the stair scenario is simulated again with enhanced difficulty. It turns out to be challenging to simulate a fluid block which reaches up until the top of the closed domain. This scenario is examined in Section 4.6.3.

4.6.1 3D Simulation of the Stair Scenario

The first scenario is a stair step scene with a fluid block placed on the top step and overlapping the step into the air. The time step δt is set to 0.5, t_{max} is increased to 350 and the three-dimensional grid size is set to $96 \times 96 \times 96$. The thresholds are set as follows: $\epsilon_{CG} = \epsilon_{Aabs} = \epsilon_{Arel} = 10^{-3}$ and $\epsilon_{Idiv} = \epsilon_{IdivBC} = 10^{-2}$.
In the middle of the right wall of the domain, a solid plane is placed such that the fluid is splashing against it from below. This scene serves well for evaluating the detaching be-



(a) $t = 0$ (b) $t = 45$ (c) $t = 68$ (d) $t = 90$

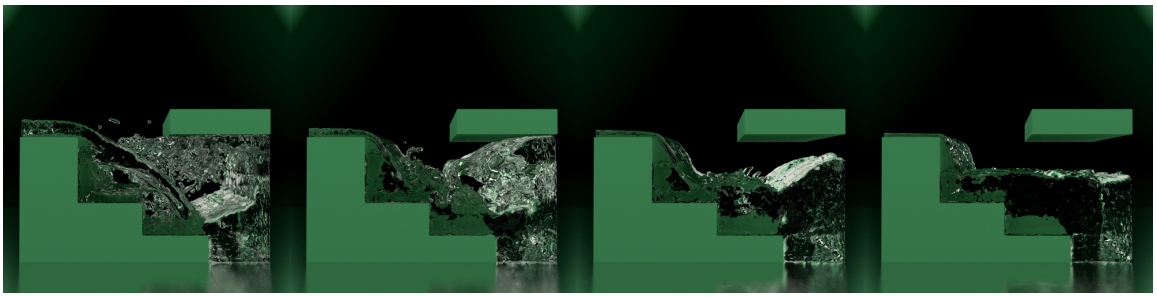


(e) $t = 133$ (f) $t = 170$ (g) $t = 225$ (h) $t = 349$

Figure 4.16: 3D Stair Scene by Original Pressure Solve

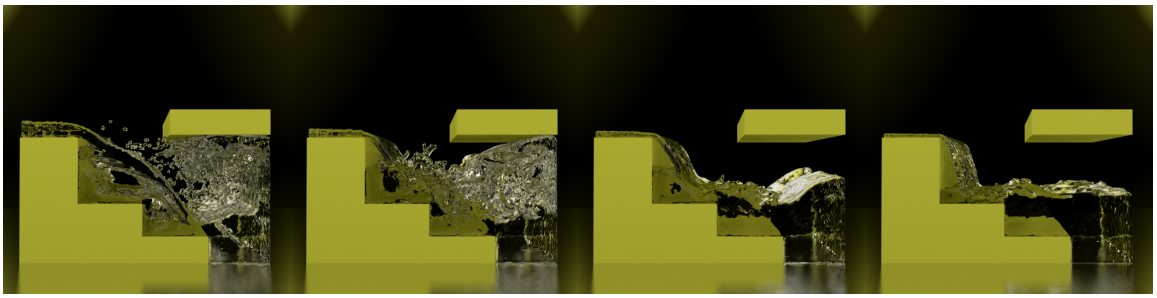
havior concerning the floating plane, by showing how fast the fluid flushes down the stairs and how much fluid will stick to the top stair. Potential artifacts can be detected where two stair steps meet. When the fluid has calmed down in the end, potential mass loss can be observed.

In Fig. 4.16, the whole stair scene is described in eight images by simulating it with OrigPS. At $t = 45$, the first part of the fluid hits the right wall to splash onto the floating

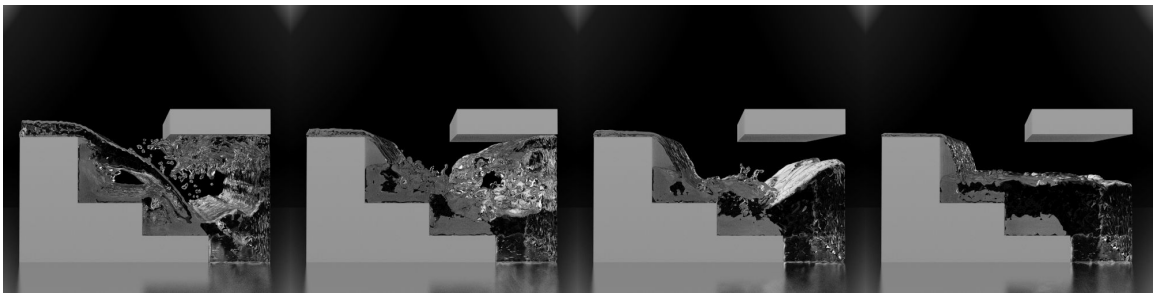


(a) OrigPSSep,$t = 90$ (b) OrigPSSep,$t = 133$ (c) OrigPSSep,$t = 170$ (d) OrigPSSep,$t = 349$

(e) ADMMVelSep,$t = 90$ (f) ADMMVelSep,$t = 133$ (g) ADMMVelSep,$t = 170$ (h) ADMMVelSep,$t = 349$



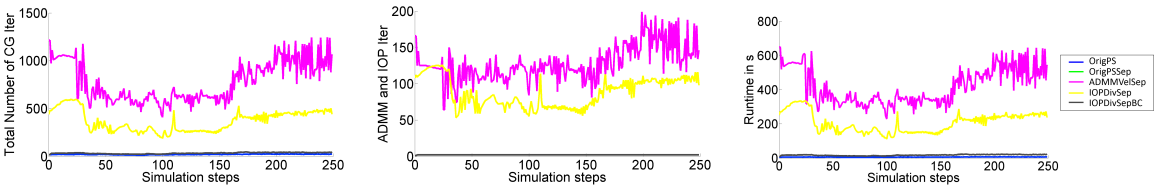
(i) IOPDivSep,$t = 90$ (j) IOPDivSep,$t = 133$ (k) IOPDivSep,$t = 170$ (l) IOPDivSep,$t = 349$



(m) IOPDivSepBC,$t = 90$ (n) IOPDivSepBC,$t = 133$ (o) IOPDivSepBC,$t = 170$ (p) IOPDivSepBC,$t = 349$

Figure 4.17: 3D Stair Scene by Original Pressure Solve

plane at $t = 68$. Around $t = 90$, the fluid should start detaching from the levitating plane and be detached completely at $t = 133$. The fluid forms a nice wave on the right wall at $t = 170$ and levels out at $t = 349$. As expected, the fluid is sticking to the solid objects when simulated with OrigPS. The fluid should detach starting from $t = 90$, but it is gluing to the levitating plane. Even at $t = 225$, many fluid drops are dangling from the plane and stick there until the end of the simulation.
In Fig. 4.17, the stair scene is simulated with OrigPS, ADMMVelSep, IOPDivSep and IOPDivSepBC. All four methods produce visually plausible results where no unnatural behavior can be identified. At time steps $t = 90$, $t = 133$ and $t = 170$, the result looks nearly identical for each method and it is impossible to examine if one of the methods produces better detaching simulations than the others. Compared to OrigPS where the fluid flows vertically down the top step at $t = 170$, the simulations with separating boundaries produce nice fluid waves flowing down the stairs at that very time step.
At $t = 349$, where the fluid has almost evened out, the fluid volume can be estimated. OrigPSSep as well as IOPDivSepBC feature slightly increased fluid volume. This might stem from the fact that for both methods, the pressure solve is applied with flags which include solid cells. However, in the classification of solid cells, a flag grid where all solid cells are exchanged by air cells is assumed. On the other hand, ADMM and IOP exhibit mass loss during the simulation.
The performance of the stair scenario is displayed in Fig. 4.18 and the corresponding averaged data is listed in Table 4.5. Although the stair scenario is more complex due to the additional obstacles, fewer number of iterations are required for ADMM and IOP compared to the breaking dam. OrigPS, OrigPSSep and IOPDivSepBC feature a similar performance. Each plot features again a characteristic curve showing which time steps consume the most computational resources. Here, time steps from $t = 0$ until $t = 40$ and $t = 160$ until the end are particularly expensive. Therefore, calm fluid is not simulated more efficiently than splashy movements.



(a) Total Number of CG Iterations (b) ADMM and IOP Iterations (c) Runtimes

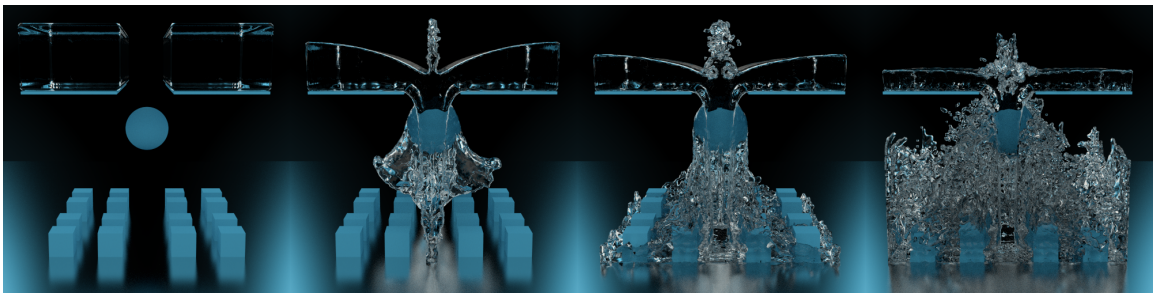
Figure 4.18: Iterations and Runtimes for the 3D Breaking Dam Simulation

Method	Color	Total CG Iter	ADMM / IOP Iter	Runtime in s
OrigPS	blue	20	-	5
OrigPSSep	green	20	-	5
ADMMVelSep	magenta	783	128	420
IOPDivSep	yellow	370	88	297
IOPDivSepBC	gray	34	1.99	17

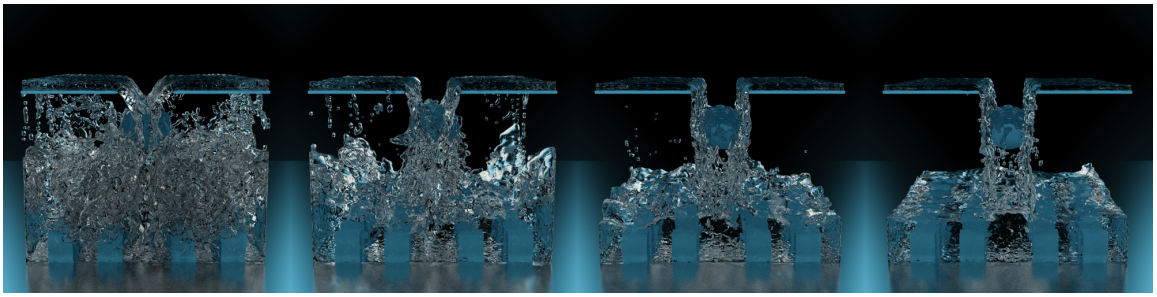
Table 4.5: Averaged Iterations and Runtimes per Time Step for 3D Stair Scene

4.6.2 3D Simulation of the Sphere Scenario

Another very challenging scenario is presented in Fig. 4.19. The simulation is applied on a $96 \times 96 \times 96$ grid with $\delta t = 0.5$, $t_{max} = 350$, $\epsilon_{CG} = \epsilon_{Aabs} = \epsilon_{Arel} = 10^{-3}$ and $\epsilon_{Idiv} = \epsilon_{IdivBC} = 10^{-2}$.
Two fluid blocks are placed onto two levitating planes on the left and right upper walls with a gap between them. Shortly below the gap, a sphere is placed on which the fluid strikes down during the simulation. 16 cubes are positioned on the floor. With this scenario, the robustness of all methods is challenged even more. The first important aspect is how the fluid is dropping down the levitating planes. Then, the fluid must detach from the sphere in a plausible way. Furthermore, the gaps between the cubes have to be filled without artifacts. The detaching behavior is once more tested when the fluid is splashing onto the levitating planes from below. In the end, the mass loss can be investigated again.
When simulating this scenario with OrigPS, the floating sphere is hit at $t = 23$. The fluid is wrapping around the sphere tightly. At $t = 45$, the fluid hits the ceiling and the floor simultaneously only to detach from the ceiling around $t = 64$. The fluid is now dispersed at the floor and flows through the aisles between the cubes towards the walls. The fluid crawls up the walls at $t = 84$ due to its acceleration and hits the sphere as well as the levitating planes from below. The fluid should detach from the planes at $t = 126$ but it sticks to them until $t = 212$. It especially stays attached to the corners when the waves are flowing high until it eventually flats out at $t = 349$. Note that the fluid is flowing down the planes vertically and wraps around the sphere. Until the end, the sphere's top is covered with fluid that does not flow down.
In Fig. 4.20, OrigPS, OrigPSSep, ADMMVelSep, IOPDivSep and IOPDivSepBC are compared. All methods with separating boundary conditions allow the fluid to detach from the ceiling earlier than OrigPS. At $t = 64$, the fluid has already detached as shown in (a),(e), (i)



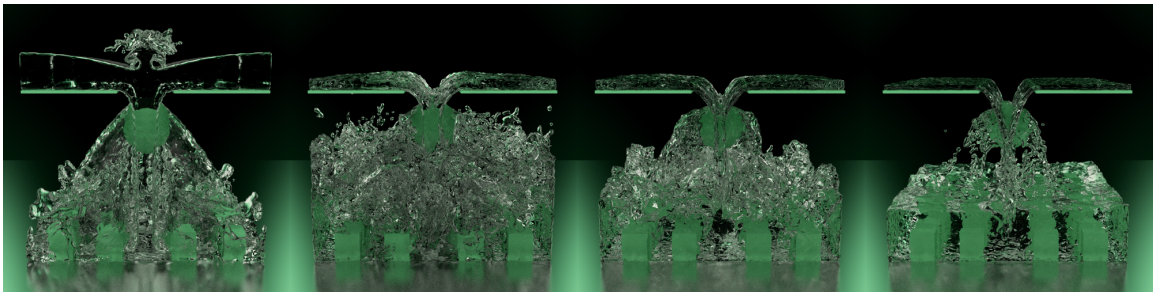
(a) $t = 0$ (b) $t = 45$ (c) $t = 64$ (d) $t = 84$



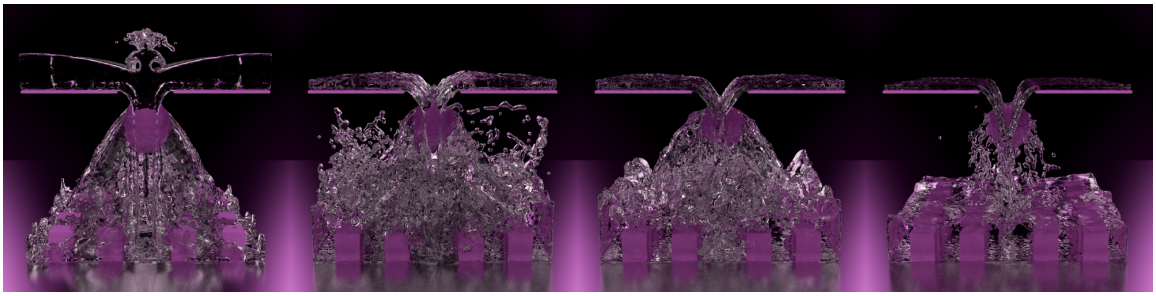
(e) $t = 126$ (f) $t = 160$ (g) $t = 212$ (h) $t = 349$

Figure 4.19: 3D Sphere Scene Simulated by the Original Pressure Solve

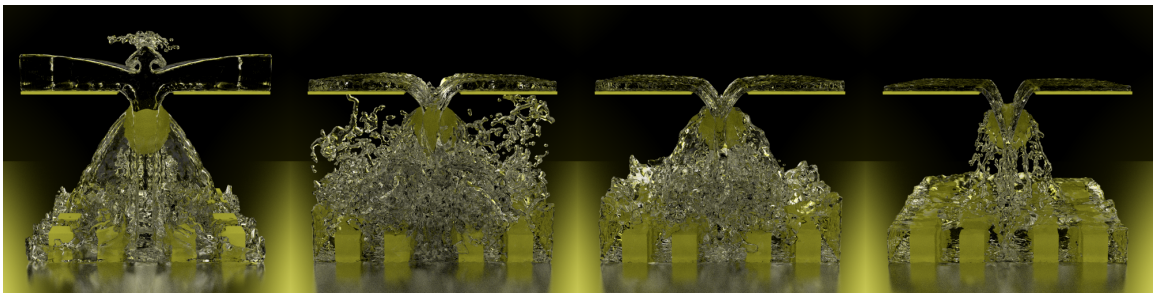
and (m). Furthermore, the fluid does not wrap around the sphere but is refracted towards the side and spread through the domain. The fluid therefore hits the walls sooner. This might also stem from the way the fluid is leaving the levitating planes. While the fluid flows down vertically in Fig. 4.19 (d), it flows down in a curved way in Fig. 4.20 (d),(h), (l) and (p) and (t). Therefore, the entrance angle of hitting the sphere is different. While the fluid is detached from the floating planes in Fig. 4.20(b),(f), (j) and (n), a big portion



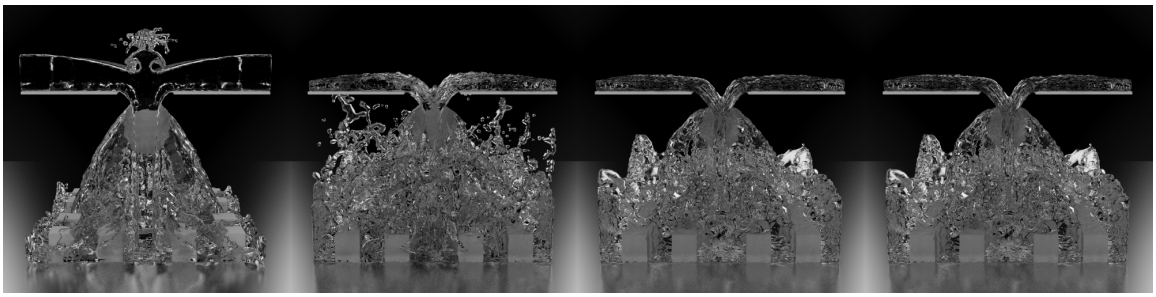
(a) OrigPSSep, $t = 64$ (b) OrigPSSep, $t = 126$ (c) OrigPSSep, $t = 160$ (d) OrigPSSep, $t = 349$



(e) ADMMVelSep, $t = 64$ (f) ADMMVelSep, $t = 126$(g) ADMMVelSep, $t = 160$(h) ADMMVelSep, $t = 349$



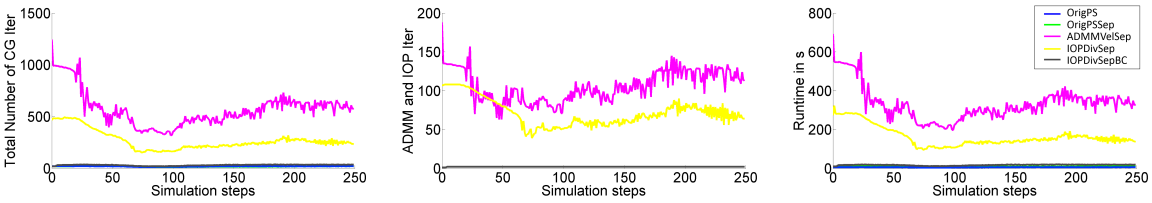
(i) IOPDivSep, $t = 64$ (j) IOPDivSep, $t = 126$ (k) IOPDivSep, $t = 160$ (l) IOPDivSep, $t = 349$



(m) IOPDivSepBC, $t = 64$ (n) IOPDivSeBC, $t = 126$ (o) IOPDivSeBC, $t = 160$ (p) IOPDivSeBC, $t = 349$

Figure 4.20: Sphere Scene by OrigPSSep, ADMMVelSep, IOPDivSep and IOPDivSeBC

of fluid is still gluing to the planes in Fig. 4.19(b), even until $t = 160$ in (c). In (c), waves press the fluid high into the corners so that it is sticking a bit too long in there in the OrigPS simulation which is not the case for OrigPSSep, ADMMVelSep and IOPDivSep. Only IOPDivSepBC shows a similar behavior but to a less extent. Compared to OrigPS, OrigPSSep shows increasing volume in $t = 349$. IOPDivSepBC provides an increased fluid volume. This might stem from allowing positive velocity at fluid-solid faces in each time step. ADMMVelSep and IOPDivSep show a slight mass loss. The mass loss might stem from the relatively low accuracies of ADMM and IOP.
Simulating the sphere scenario leads to lower iteration numbers compared to the breaking dam, similar as for the stair scene. The most challenging frames are again in the beginning and the end of the simulation, see Fig. 4.21. This time, even OrigPS, OrigPSSep and IOPDivSepBC need fewer iterations simulating the sphere scenario as shown in Table 4.6.



(a) Total Number of CG Iterations (b) ADMM and IOP Iterations (c) Runtimes

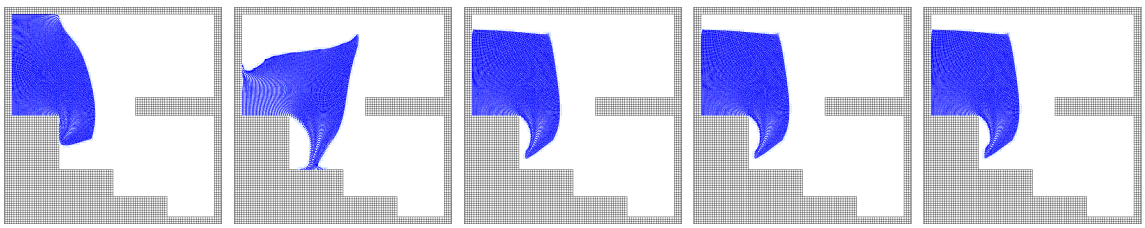
Figure 4.21: Iterations and Runtimes for the 3D Breaking Dam Simulation

Method	Color	Total CG Iter	ADMM / IOP Iter	Runtime in s
OrigPS	blue	18	-	5
OrigPSSep	green	16	-	9
ADMMVelSep	magenta	569	106	311
IOPDivSep	yellow	269	71	164
IOPDivSepBC	gray	31	1.98	17

Table 4.6: Averaged Iterations and Runtimes per Time Step for 3D Sphere Scene

4.6.3 3D Simulation of the Stair Scenario with Increased Difficulty

When increasing the height of the fluid block in the stair scene in Fig. 4.16 until the fluid touches the ceiling, the robustness of all methods is challenged heavily as shown in the 2D simulation in Fig. 4.22. The results in Fig. 4.22 are run with the practical setting of



(a) OrigPS (b) OrigPSSep (c) ADMMVelSep (d) IOPDivSep (e) IOPDivSepBC

Figure 4.22: 2D Stair Scene with Fluid Touching the Ceiling at $t = 20$

$\epsilon_{CG} = \epsilon_{Idiv} = \epsilon_{Aabs} = \epsilon_{Arel} = 10^{-3}$ and $\epsilon_{Idiv} = 10^{-2}$.
When simulating the scene with OrigPS in Fig. 4.22 (a), the fluid is gluing unnaturally to the stair steps and the ceiling. On the other hand, OrigPSSep in Fig. 4.22 (b) shows how the introduced divergence can effect the whole simulation at its worst case scenario. The pressure solve is calculated on the basis that all the ceiling cells are classified as solids. Later on, in the pressure update, those cells are treated as air cells since the fluid is currently detaching. The velocity is allowed to be positive in the direction of the surface normal. But apparently, the fluid block experiences a huge acceleration away from the left wall and ceiling. Fortunately, ADMMVelSep, IOPDivSep and IOPDivSepBC in Fig. 4.22 (c), (d) and (e) provide much better results. The fluid is neither sticking to the ceiling, nor gluing to the left wall. The fluid forms a nice wave flowing down the stairs. The only slight visual peculiarities are at the upper corners of the fluid block: the left corner is detaching a little from the left wall and the right corner is lifted up a bit. This might also stem from the forces dragging the fluid down the stairs.
While both OrigPS and OrigPSSep are visually implausible, ADMM, IOP and the accelerated method IOPDivSepBC produce nice results. Increasing the accuracies of OrigPS and OrigPSSep does not change the dominant fluid motion. Classifying solid cells based on the pressure does not affect the fluid motion in OrigPSSep either.

Chapter 5

Conclusion and Outlook

Despite the high demand for fast and visually plausible fluid simulations, common fluid solvers feature often insufficiently separating solid wall boundary conditions. This results in visually implausible fluid behavior where the fluid is sticking unnaturally long to solid objects.
The goal of this thesis was hence to enable flexible solid wall boundary conditions for simple pressure solvers. The main contribution was to investigate multiple approaches, especially a method based on proximal operators (ADMM), to provide separating solid wall boundary conditions based on a normal CG pressure solver.

5.1 Summary and Discussion

After giving a thorough overview on related work in Section 1.3, the basics of fluid simulation, ADMM and IOP were discussed in Chapter 2. As a next step, the implementation of ADMM and IOP as pressure solve was explained and elaborated in Chapter 3. One of the difficulties was to split the original pressure solve properly into the two sub-problems, i.e. solving the incompressibility conditions and ensuring the boundary conditions. Furthermore, separating boundary conditions were implemented in the original CG solver which unfortunately introduces divergence at fluid-solid faces. To profit from the accuracy of the ADMM and IOP implementations and the efficient adjusted CG solver, a hybrid method was developed.
In Chapter 4, the four implemented methods were extensively evaluated, discussed and compared.

Against other predictions in [Bri08], it is possible to implement separating solid wall boundary conditions within a simple CG solver when classifying solid cells into free-surface and solid walls after the pressure solve and using this classification in the pressure update. Borders between fluid and solid cells, which were classified as free-surface, are allowed to have positive normal velocity components while the normal velocity component is set to zero otherwise. Since the pressure equations are solved before the classification, the pressure update operates on a different solid cell classification which introduces divergence at the fluid cells next to solid cells classified as free-surface. However, the divergence can be eliminated by applying the CG pressure solve several times until the desired level of divergence is undercut. The classification of solid cells is applied only after the first pressure solve to avoid changing the boundary conditions in each iteration which would cause jittering artifacts. The performance is very good, solely twice as many iterations are carried out and the pressure solve takes twice as long. In fact, the adjusted CG pressure solve needs 1.8 times as many iterations and runtime as the original CG pressure solve. Note that one simulation step consists of an advection step, applying body forces and the pressure solve. The performance of the fluid simulation besides the pressure solve remains the same. When choosing a rather less challenging scenario, both variants provide visually very plausible results.
The classification of the solid cells can be conducted in two different ways. The classification in this work is based on examining the resulting normal velocity components. Assuming

that all solid cells are classified as free-surface and therefore treated as empty air cells, a hypothetical pressure update is applied to the velocity field. Those cells with positive normal velocity components are classified as free-surface. The reason for the classification is based on the velocity in this thesis, is the further goal to replace the CG pressure solver with a more efficient method that makes the velocity field divergence-free. The pressure is not necessarily calculated. The classification can also be conducted by evaluating the pressure. Negative pressure in a fluid cell next to a solid cell indicates that the fluid is about to detach from the solid wall. Therefore, those solid cells are classified as free-surface. The comparison of both classification methods show that classifying by means of the pressure leads to slightly less detaching fluid but overall to the same fluid behavior.
When the classification is conducted based on the velocity directly before the pressure solve, which would avoid different roles of solid cells in the pressure solve and pressure update, too many cells are classified as free-surface which in turn leads to tremendous mass loss due to semi-permeable boundaries.

Splitting the pressure solve into two sub-problems, making the velocity field divergence-free and ensuring the solid wall boundary conditions, provides a suitable case for applying ADMM. The incompressibility condition is met by applying the CG pressure solver. Handling the boundary conditions means to set the velocity at fluid-solid faces to zero where the solid cells are classified as solid wall or where the normal velocity component is below zero. The simulations with ADMM provide visually plausible results and are perfectly robust to changes in the scene topology, e.g. when adding obstacles. Unfortunately, the performance is quite poor. ADMM needs about 38 times more CG iterations per time step than an original CG pressure solve for accuracies in the range of 10^{-3}.
Another method for implementing separating solid wall boundaries is Iterated Orthogonal Projection (IOP) which is related to ADMM. Instead of solving both sub-problems on two different velocity fields and connecting them via a dual variable update, IOP operates on only one velocity field. The resulting simulations are equally visually plausible and robust, but the performance is slightly better as far as both methods are comparable. The total number of CG iterations per time step is approximately 20 times higher than for the original CG pressure solver for accuracies in the range of 10^{-3}.
CG, ADMM and IOP do not share a common stopping criterion. Choosing the divergence as stopping criterion works well for CG and IOP, but not for ADMM. This is due to the fact that the residual for ADMM decreases in a periodic way and since the divergence is only measuring how well the incompressibility condition is met, stopping at a minima of the divergence means stopping when the boundary conditions are poorly met. The usual residual in ADMM is calculating the difference between both velocity fields and it is phase shifted to the divergence. The difference of two velocity fields cannot be calculated for IOP or CG due to the missing second velocity field. Using a velocity field from the previous iteration leads to measuring a different change.
Simulations with ADMM and IOP converge to an agreement solution, their difference lies in the range of the chosen accuracy. Adjusting the CG solver as mentioned at the beginning of this section does not lead to the same agreement solution which means that the solved problem is a different one.

5.2 Outlook

The adjusted CG solver already leads to efficient and visually appealing results. Yet, harnessing ADMM or IOP to simulate fluid comes with high resource consumption. Since information about solid boundary conditions is not passed to the CG pressure solve, the pressure solve can be conducted by very efficient solvers that in turn cannot handle solid wall boundary conditions very well. For example, making the velocity field divergence-free can be solved in the frequency domain very efficiently if periodic boundary conditions are provided which is the case here. As stated in [Hen12], an FFT solver could solve the discrete Poisson equation with N total grid points with a serial complexity of $O(N \log N)$ and parallel complexity of $O(\log N)$ time while a CG solver is in $O(N^{3/2})$ or $O(N^{1/2} \log N)$ respectively. Therefore, the pressure solve could be exchanged by a FFT solver which would improve the performance of IOP and ADMM heavily.
Furthermore, the practical parameter settings for IOP and ADMM should be analyzed independently from each other. IOP provides visually plausible results for lower accuracies than the chosen setting of $\epsilon_{CG} = 10^{-3}$ and $\epsilon_{Idiv} = 10^{-2}$. But in order to achieve comparable simulations with both methods, only parameter settings convenient for both methods were considered. In contrast to IOP, ADMM can also be conducted by choosing a parameter setting where ϵ_{CG} is less strict then the accuracy for ADMM.
Additionally, further stopping criteria could be analyzed to speed up convergence.
Besides that, examining the adjusted CG solver should be of high priority since it already produces great results in high performance. The possibility to determine detaching cells before the pressure solve should be investigated, such that both the pressure solve and the pressure update operate on the same cell classification.
High value of research lies in the analysis of the behavior of all methods for the scene where the fluid box touches the top of the domain. Since the original as well as the adjusted pressure solve show visually implausible results, further investigation about this erratic behavior and possible solutions is required.

Bibliography

[BBB07] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph.*, 26(3), July 2007.

[Boy11] Stephen Boyd. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[Bri08] Robert Bridson. *Fluid Simulation for Computer Graphics*. A K Peters/CRC Press, September 2008.

[BTT09] M. Becker, H. Tessendorf, and M. Teschner. Direct forcing for lagrangian rigid-fluid coupling. *Visualization and Computer Graphics, IEEE Transactions on*, 15(3):493–503, May 2009.

[CM11] Nuttapong Chentanez and Matthias Mueller. A multigrid fluid pressure solver handling separating solid boundary conditions. pages 83–90, 2011.

[CMT04] Mark Carlson, Peter J. Mucha, and Greg Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 377–384, New York, NY, USA, 2004. ACM.

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*, pages 61–76, New York, NY, USA, 1996. Springer-Verlag New York, Inc.

[EF02] Doug Enright and Ron Fedkiw. Robust treatment of interfaces for fluid flows and computer graphics. In *Computer Graphics, 9th Int. Conf. on Hyperbolic Problems Theory, Numerics, Applications*, 2002.

[Erl13] Kenny Erleben. Numerical methods for linear complementarity problems in physics-based animation. In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, pages 8:1–8:42, New York, NY, USA, 2013. ACM.

[FF01] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 23–30, New York, NY, USA, 2001. ACM.

[FM96] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graph. Models Image Process.*, 58(5):471–483, 1996.

[FSJ01] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 15–22, New York, NY, USA, 2001. ACM.

[GB13] Dan Gerszewski and Adam W. Bargteil. Physics-based animation of large-scale splashing liquids. *ACM Trans. Graph.*, 32(6):185:1–185:6, November 2013.

[GHD13] Olivier Gnevaux, Arash Habibi, and Jean-Michel Dischler. Simulating fluid-solid interaction. *Graphics Interface*, pages 31–38, June 2013.

[GITH14] James Gregson, Ivo Ihrke, Nils Thuerey, and Wolfgang Heidrich. From capture to simulation: connecting forward and inverse problems in fluids. *ACM Trans. Graph.*, 33(4):1–11, 2014.

[Har63] Francis H. Harlow. The particle-in-cell method for numerical solution of problems in fluid dynamics. *Experimental arithmetic, high-speed computations and mathematics*, 1963.

[Hen12] Ronald D. Henderson. Scalable fluid simulation in linear time on shared memory multiprocessors. In *Proceedings of the Digital Production Symposium*, DigiPro '12, pages 43–52, New York, NY, USA, 2012. ACM.

[HW65] Francis H. Harlow and J. Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids (1958-1988)*, 8(12):2182–2189, 1965.

[KLLR05] Byungmoon Kim, Yingjie Liu, Ignacio Llamas, and Jaroslaw R. Rossignac. Flowfixer: Using bfecc for fluid simulation. Technical report, Georgia Institute of Technology, 2005.

[KTJG08] Theodore Kim, Nils Thuerey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. *ACM Trans. Graph.*, 27(3):1–6, 2008.

[Lap03] Soleil Lapierre. An investigation of fourier domain fluid simulation, 2003.

[LGF04] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 457–462, New York, NY, USA, 2004. ACM.

[man14] mantaflow - an extensible framework for fluid simulation. http://mantaflow.com/, 2014.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[MCPN08] Jeroen Molemaker, Jonathan M. Cohen, Sanjit Patel, and Jonyong Noh. Low viscosity flow simulations for animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 9–18, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[MSJT08] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. Real time physics: Class notes. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 88:1–88:90, New York, NY, USA, 2008. ACM.

[NGF02] Duc Nguyen, Frdric Gibou, and Ronald Fedkiw. A fully conservative ghost fluid method stiff detonation waves. In *In 12th Int. Detonation Symposium*, 2002.

[PB13] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123231, 2013.

[REN+04] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '04, pages 193–202, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[SB12] Hagit Schechter and Robert Bridson. Ghost sph for animating water. *ACM Trans. Graph.*, 31(4):61:1–61:8, July 2012.

[Sca14] Scanline VFX. Showreel of 2012. http://www.scanlinevfx.com/la/en/reels.html, 2014. Accessed: 2014-10-30.

[SFK+08] Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable maccormack method. *J. Sci. Comput.*, 35(2-3):350–371, June 2008.

[SPDC10] Mario A. Storti, Rodrigo R. Paz, Lisandro D. Dalcin, and Santiago D. Costrarelli. A fft preconditioning technique for the solution of incompressible flow with fractional step methods on graphic processing units. *Mecnica Computacional*, 29(0):7123–7145, 2010.

[Sta99] Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[Sta01] Jos Stam. A simple fluid solver based on the fft. *Journal of Graphics Tools*, 6(2):43–52, 2001.

[TFK+03] Tsunemi Takahashi, Hiroko Fujii, Atsushi Kunimatsu, Kazuhiro Hiwada, Takahiro Saito, Ken Tanaka, and Heihachi Ueki. Realistic animation of fluid with splash and foam. *Computer Graphics Forum*, 22(3):391–400, 2003.

[WBAW12] Bo Wahlberg, Stephen Boyd, Mariette Annergren, and Yang Wang. An admm algorithm for a class of total variation regularized estimation problems. In *16th IFAC Symposium on System Identification*, 2012.

[ZB05] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, 2005.