Fast and Scalable Solvers for the Fluid Pressure Equations with Separating Solid Boundary Conditions

by

Junyu Lai

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We propose and evaluate fast, scalable approaches for solving the linear complementarity problems (LCP) arising from the fluid pressure equations with separating solid boundary conditions. Specifically, we present a policy iteration method, a penalty method, and a modified multigrid method, and demonstrate that each is able to properly handle the desired boundary conditions. Moreover, we compare our proposed methods against existing approaches and show that our solvers are more efficient and exhibit better scaling behavior; that is, the number of iterations required for convergence is essentially independent of grid resolution, and thus they are faster at larger grid resolutions. For example, on a 256^3 grid our multigrid method was 30 times faster than the prior multigrid method in the literature.

Acknowledgements

First and foremost, I must thank my supervisor, Justin Wan, for his invaluable tutoring, advice and perspective. He guided me through everything from research ideas to theoretic fundamentals and coding techniques. His interest and enthusiasm in my work made completing this thesis an interesting and rewarding process.

I would like to thank my readers, Christopher Batty and Kimon Fountoulakis. Their insight and thoughtfulness provided important contributions to the process and product. Special thanks also to Prof Batty for his invaluable assistance during my graduate studies.

Finally I would like to thank my family for their encouragement and understanding.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Motivated by the demand for realistic visual effects in the film and game industry, fluid animation has been explored in the computer graphics community for many years. Fluids are ubiquitous and indispensable in our daily life: the water from the sea, the coffee or tea in our cups, the blood running through our veins, the gasoline for fueling vehicles, and more. People see and interact with real fluids every day and expect to experience the same effects when they appear on the screen. Getting used to how fluids behave naturally, even minor inaccuracies in the animation may bring discomfort to the audiences. It comes with no surprise that fluid simulation is an important topic in computer graphics.

Due to complexity of fluid dynamics, it is difficult to animate fluid effects frame by frame based on visual appearance [56]. Therefore, physical based methods are widely used for simulating realistic fluids by solving the equations derived from physics. The Navier-Stokes equations are commonly used to describe the fluid flows of interest in simulations. Solving the Navier-Stokes equations consists of the following steps [11]: advecting the fluid and its velocities through the flow; applying body forces such as gravity; applying viscosity if dealing with viscous fluid; and performing pressure projection to enforce incompressibility.

One obstacle to the realistic fluid simulation is the mishandling of the fluid near the solid boundary. Standard solid boundary conditions do not allow fluid to naturally separate from a solid boundary; instead, the inviscid fluid unnaturally adheres to the top and side walls of a domain. Such unrealistic effects are usually noticeable in fluid simulations. To resolve this issue, Batty et al. [8] proposed a new free surface/solid boundary condition for the fluid-solid wall that allows separation while disallowing penetration. However, while this corrects the behavior, it transforms the pressure equation from a standard linear system into a linear complementarity problem (LCP) which is even more challenging to

solve efficiently. As a result, this improved boundary condition has seldom been adopted in practice.

Solving the pressure equation often comprises a significant fraction of the simulation time [18], especially when the problem sizes are large. Therefore, it is important to develop a fast and scalable numerical approach. We consider a method to be scalable if the number of iterations is essentially independent of the mesh resolution. When the pressure equation is a linear system, the preconditioned conjugate gradient (PCG) algorithm is commonly used in the computer graphics community. However, PCG is not scalable as the number of iterations increases linearly with the mesh resolution. With the modified solid boundary conditions, the pressure equation becomes a LCP problem and cannot even be solved with PCG, which presents more challenges to simulating realistic fluid efficiently.

Several approaches are used to solve LCP problems in fluid simulations. One way is to formulate the LCP problem into a convex optimization problem. [26, 50, 34, 40]. Newton method [3, 25] is also proposed and modified to solve the LCP and performs better than projected Gauss-Seidel (PGS) type methods [23, 17, 33] and the pivoting methods [5, 2]. These methods usually involve solving several linear systems. It is not clear how scalable they are in terms of the number of linear systems and the number of iterations required for solving a linear system. Mutigrid is widely known for its good scalability. Various multigrid schemes [49, 48, 27] have been used for fluid simulation with standard solid boundary conditions and achieved scalable performance, but these require solving only linear systems rather than LCPs. Chentanez & Muller [14] developed a multigrid method to solve the LCPs but the scalability is not clear yet.

To resolve the issues mentioned above, we propose more efficient and scalable solvers to speed up realistic fluid simulations. Specifically, we develop and evaluate variants of the policy iteration [30], the penalty method [20], and the full approximation scheme multigrid (FAS-MG) [37] for solving LCPs arising from the separating boundary conditions, because such schemes are known to be convergent and efficient. While numerical schemes belonging to these families of methods have been explored for problems arising in computational finance [30, 20, 37], to our knowledge we are the first to consider their use in the context of fluid animation, or computer animation more broadly. Our results show that our proposed methods are both more scalable and more efficient compared with existing approaches.

The rest of the thesis is organized as follows. At first, we elaborate the previous work in Chapter 2. In Chapter 3, we introduce the Navier-Stokes equations and boundary conditions, discuss the standard fluid simulation algorithm, and describe the numerical methods in fluid simulation. We review the definition of the LCP problem and illustrate how it is formulated using the separating solid boundary conditions in Chapter 4. The main

contribution of this thesis, namely the proposed fast and scalable solvers: policy iteration, penalty method, full approximation scheme multigrid, are described in Chapter 5. We present the numerical results including comparison with recent LCP solvers in Chapter 6. We conclude the thesis and discuss potential extension of our methods to rigid body simulations in Chapter 7.

Chapter 2

Previous work

We summarize prior work in solving LCPs from fluid simulations or related fields (eg. rigid body simulations) in computer graphics and present the motivation of our proposed methods. The LCP problem adds constraints to the linear problem to make the linear component complementary to the constraints. This gives a nonlinear problem, which cannot be solved with the existing linear solvers. Solving the LCP problem requires locating the exact complementary positions to enforce the constraints in order to make it linear. This is hard and computationally expensive especially for large problems. It is also desirable to convert the LCP into sub-steps which are linear problems. However, it is difficult to achieve good overall convergence and scalability when the problem is large.

One way to solve the LCP is to exploit its combinatorial nature. A direct approach called pivoting method is proposed for solving LCP problems. The idea is to incrementally find all the complementary entries to enforce the constraints. The use of pivoting method can be traced in the rigid body simulation [5, 47], and collision detection [44]. Although pivoting methods can provide accurate solutions to LCPs, they may suffer from exponential time complexity in the worst cases, which is not computationally efficient [25].

In contrast with the direct approaches, iterative methods are used for solving LCPs through iteratively improving the solutions. One such method is the projected Gauss-Seidel (PGS) method which is based on the Gauss-Seidel method for solving linear systems. The idea is to enforce the constraints from the LCP after each Gauss-Seidel iteration. It has been used in several applications in computer graphics simulations including shock propagation for multibody animation [23], deformable body contact [17], and adhesive contact [33]. However, as pointed out in [33], PGS may suffer from slow convergence and it is desirable to find more efficient LCP solvers.

The LCP can also be solved by converting it to a convex optimization problem, for example, quadratic programming (QP), and solved using optimization techniques. The PATH solver [26], which is a generalization of the classical Newton method and based on quadratic programming (QP), was used by Batty et al. [8] to solve the LCP problem. However, they point out that it is not scalable to large problems. Narain et al. [50] formulated a pressure equation for granular material simulation into the LCP. Gerszewski et al. [34] solved LCPs for pressure and density when animating large-scale splashing liquids. Both Narain et al. [50] and Gerszewski et al. [34] used a QP solver called modified proportioning with reduced gradient projections (MPRGP) [19] to solve LCPs. MPRGP is an active set method based on preconditioned conjugate gradient (PCG) that interleaves conjugate gradient (CG) steps with the update of the active set. Inglis et al. [40] proposed a Primal-Dual method to split the convex optimization problem into two components and solved them with CG and a classification scheme, respectively.

In addition to the convex optimization problem, Andersen et al. [3, 25] formulated the LCPs in fluid animation in 2D into minimum map equations, which are nonsmooth and nonlinear. They proposed a nonsmooth Newton approach to solve these minimum map equations. They modify the standard Newton method and perform Newton iterations until convergence. Their method has better convergence than projected Gauss–Seidel (PGS) type methods [23, 17, 33] and is faster than pivoting methods [5, 2], but requires solving a linear system and performing line searches on each Newton iteration. A drawback of their method is that the matrix from each Newton iteration may be singular. It is observed that the overall solver fails in some cases when using the preconditioned conjugate gradient (PCG) as the linear solver. Therefore, the standard conjugate gradient (CG) method, which is less efficient, has to be used. Andersen et al.[3] extend their framework to 3D [4] and demonstrate convergence for a 100^3 grid. However, they did not discuss how their method scales with larger grid resolution.

For both optimization approach and Newton method, either PCG or CG is used as the linear solver. However, PCG and CG are not scalable as the number of iterations is known to double when grid resolution is doubled along each dimension, i.e., when the width of each cell is halved [46]. The multigrid method is a solution to resolve the scalability issue. The idea of multigrid method is to eliminate high frequency errors on the fine grid and reduce low frequency errors on the coarse grid, which is less expensive. Chentanez & Muller [14] developed a multigrid method to solve the LCPs from fluid simulations. Their method requires only a few small changes to multigrid for linear systems [15]. They apply these changes only to the finest three grids and do the same thing as the linear multigrid does on the rest of grids. Their method converges for several scenarios in both 2D and 3D. However, they did not perform any scaling tests for large problems to demonstrate

whether it achieves mesh-independent convergence behavior, which is the major advantage of using a multigrid scheme.

Although the recent methods addressed the convergence issues for solving LCPs in fluid simulations, the question of scalability still remains open. Thus, we propose fast and scalable LCP solvers suitable for large scale fluid simulations and demonstrate improvements upon the existing approaches in terms of both convergence and scalability.

Chapter 3

Fluid simulation

In this chapter, we will discuss how to simulate incompressible fluid in computer graphics. First, we describe the derivation of incompressible Navier-Stokes equations. Since we only deal with inviscid fluid only, we explain how to drop the viscosity term and obtain the incompressible Euler equations. Then we discuss boundary conditions and show how the complementary equations are formulated from separating solid boundary conditions. We will introduce the standard simulation algorithm, which decomposes the incompressible Euler equations into three sub equations. Finally, we present the numerical methods for solving these equations.

3.1 Notation

Before introducing the equations and formulas in fluid simulation, we first introduce some notations. Let $\mathbf{x} = (x_1, x_2, x_3) = (x, y, z)$ be the position vector. We denote $\mathbf{u}$ as the fluid velocity vector, t as the time, T as the termination time, p as the pressure, ρ as the density, ν as the kinematic viscosity coefficient, and $\mathbf{g}$ as the acceleration due to body forces such as gravity. The expression of the velocity vector is given by $\mathbf{u} = (u_1, u_2, u_3) = (u, v, w)$ where $u = u(\mathbf{x}, t)$, $v = v(\mathbf{x}, t)$, $w = w(\mathbf{x}, t)$ are the velocities in the x, y, z directions at time t, respectively. The pressure $p = p(\mathbf{x}, t)$ is a scalar representing the force applied per unit area at the position $\mathbf{x}$ and the time t. We define the gradient operator as $\boldsymbol{\nabla} = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$, and the divergence operator as $\boldsymbol{\nabla}\cdot = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})\cdot$. The advection operator $\mathbf{u} \cdot \boldsymbol{\nabla}$ is defined as $u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + w\frac{\partial}{\partial z}$. The notations in 2D can be defined similarly.

3.2 Navier-Stokes equations

In computer graphics, fluid simulations are usually modelled by the incompressible Navier-Stokes equations:

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \boldsymbol{\nabla})\mathbf{u} + \frac{1}{\rho}\boldsymbol{\nabla} p = \nu \Delta \mathbf{u} + \mathbf{g} \\ \\ \boldsymbol{\nabla} \cdot \mathbf{u} = 0. \end{cases} \tag{3.1}$$

We proceed to describing the derivation of the above equations using the incompressibility, the law of mass conservation and the law of momentum conservation [35]. Consider a chunk of fluid occupying an arbitrary domain Ω_t at time t with volume of V_t, we denote its surface as $S_t = \partial V_t$. For a differentiable scalar function $f : \Omega_t \times [0, T] \to \mathbb{R}$, the transport theorem gives:

$$\frac{d}{dt}\int_{\Omega_t} f(\mathbf{x}, t) d\mathbf{x} = \int_{\Omega_t} \big\{\frac{\partial}{\partial t} f + \boldsymbol{\nabla} \cdot (f\mathbf{u})\big\}(\mathbf{x}, t) d\mathbf{x}. \tag{3.2}$$

The mass of the fluid occupying Ω_t is calculated by integrating over the fluid density $\rho(\mathbf{x}, t)$:

$$\int_{\Omega_t} \rho(\mathbf{x}, t) d\mathbf{x}. \tag{3.3}$$

Since the mass of this chunk of fluid remain constant over time, the derivative of mass with respect to time becomes zero. Letting $f = \rho$, the transport theorem yields

$$\int_{\Omega_t} \big\{\frac{\partial}{\partial t} \rho + \boldsymbol{\nabla} \cdot (\rho\mathbf{u})\big\}(\mathbf{x}, t) d\mathbf{x} = 0, \tag{3.4}$$

which holds for arbitrary domain Ω_t. Therefore, the integrand vanishes and we obtain the following equation:

$$\frac{\partial}{\partial t} \rho + \boldsymbol{\nabla} \cdot (\rho\mathbf{u}) = 0. \tag{3.5}$$

Since we are dealing with incompressible flow, the constant density ρ gives:

$$\boldsymbol{\nabla} \cdot \mathbf{u} = 0, \tag{3.6}$$

which is exactly the second equation in (3.1).

Now we proceed to the derivation of the first equation in (3.1) based on the conservation of momentum. The momentum of the fluid in Ω_t is expressed by integrating the product of the mass with the velocity:

$$\int_{\Omega_t} \rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t) d\mathbf{x}. \tag{3.7}$$

By Newton's second law, the change of momentum equals to the sum of forces which include the body forces, surface forces, and internal friction. We express the body forces as

$$\int_{\Omega_t} \rho(\mathbf{x},t)\mathbf{g}(\mathbf{x},t)d\mathbf{x}, \tag{3.8}$$

and the surface forces as

$$\int_{S_t} \boldsymbol{\sigma}(\mathbf{x},t)\mathbf{n}dS, \tag{3.9}$$

where $\boldsymbol{\sigma}$ is the stress tensor, $\mathbf{n}$ is the normal component of the velocity around the surface S_t. According to Newton's law, we have

$$\frac{d}{dt}\int_{\Omega_t} \rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t)d\mathbf{x} = \int_{\Omega_t} \rho(\mathbf{x},t)\mathbf{g}(\mathbf{x},t)d\mathbf{x} + \int_{S_t} \boldsymbol{\sigma}(\mathbf{x},t)\mathbf{n}dS. \tag{3.10}$$

Applying the transport theorem (3.2) to the left hand side of (3.10) and the divergence theorem to the first term on the right hand side, we obtain the equation for the momentum:

$$\frac{\partial}{\partial t}(\rho\mathbf{u}) + (\mathbf{u}\cdot\boldsymbol{\nabla})(\rho\mathbf{u}) + (\rho\mathbf{u})\boldsymbol{\nabla}\cdot\mathbf{u} - \rho\mathbf{g} - \boldsymbol{\nabla}\cdot\boldsymbol{\sigma} = 0. \tag{3.11}$$

For Newtonian fluids obeying the Stokes assumption the stress tensor $\boldsymbol{\sigma}$ can be modelled as follows:

$$\boldsymbol{\sigma} := -p\mathbf{I} + \boldsymbol{\tau} := (-p + \lambda\boldsymbol{\nabla}\cdot\mathbf{u})\mathbf{I} + 2\mu\boldsymbol{\delta}, \tag{3.12}$$

where $\boldsymbol{\tau}$ is the viscous part, λ and μ are thermodynamic material constants and $\boldsymbol{\delta}$ is the strain tensor defined as

$$\boldsymbol{\delta} := \frac{1}{2}\Big[\Big(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\Big)\Big]_{i,j=1,2,3}. \tag{3.13}$$

Substituting (3.12) and (3.13) into the momentum equation (3.11) yields

$$\frac{\partial}{\partial t}(\rho\mathbf{u}) + (\mathbf{u}\cdot\boldsymbol{\nabla})(\rho\mathbf{u}) + (\rho\mathbf{u})\boldsymbol{\nabla}\cdot\mathbf{u} + \boldsymbol{\nabla}p = (\mu+\lambda)\boldsymbol{\nabla}(\boldsymbol{\nabla}\cdot\mathbf{u}) + \mu\Delta\mathbf{u} + \rho\mathbf{g}. \tag{3.14}$$

Since we assume the flow is incompressible (ρ is constant), applying the equation (3.6) gives

$$\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{u} + \frac{1}{\rho}\boldsymbol{\nabla}p = \frac{\mu}{\rho}\Delta\mathbf{u} + \mathbf{g}, \tag{3.15}$$

which is the first equation of (3.1) if we define the kinematic viscosity as $\nu = \frac{\mu}{\rho}$.

As we are interested in dealing with inviscid fluids, we drop the viscosity part $\boldsymbol{\tau}$ in (3.12), which will consequently drop the viscosity term $\frac{\mu}{\rho}\Delta\mathbf{u}$ in (3.15) and present the incompressible Euler equations as follows:

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{u} + \frac{1}{\rho}\boldsymbol{\nabla} p = \mathbf{g} \\ \\ \boldsymbol{\nabla}\cdot\mathbf{u} = 0. \end{cases} \tag{3.16}$$

3.3 Boundary conditions

So far we have only described how the Navier-Stokes equations are used to model the dynamics of the fluid inside the computational domain. However, we still need to specify the fluid behaviour on the surface boundary. Typically, it is enforced by setting the appropriate boundary conditions. In this section, we will present the standard boundary conditions commonly used in computer graphics, followed by a modified condition that is able to better capture the fluid-solid motions. We illustrate the issue with the standard boundary conditions at the fluid-solid interface and how it can be resolved with separating solid boundary conditions.

3.3.1 Standard boundary conditions

At the interface between the fluid and air, we assume there is no force from the air to the fluid. Hence we set the pressure to be zero by imposing the following Dirichlet boundary condition:

$$p = 0 \tag{3.17}$$

at the free surface between fluid and air.

Let's now consider the solid wall boundary. Let $\mathbf{u}_{solid}$ be the solid velocity and $\mathbf{n}$ be the outward normal. The standard solid wall boundary conditions are given as

$$\mathbf{u}\cdot\mathbf{n} = \mathbf{u}_{solid}\cdot\mathbf{n}, \tag{3.18}$$

which prevents fluid from crossing solid boundary.

3.3.2 Separating solid boundary conditions

For simplicity, we assume the solid is static, namely $\mathbf{u}_{solid} = \mathbf{0}$. The standard solid boundary conditions, $\mathbf{u} \cdot \mathbf{n} = 0$, prevent fluid from flowing into or out of a solid but have the side-effect of not allowing fluid to separate from a solid [8]. As a result, the fluid sticks to the solid walls as shown in Figure 3.1 (Left). To allow the fluid to separate from the walls but not flow into it, we instead model the solid wall boundary conditions as follows:

$$0 \leq p \perp \mathbf{u} \cdot \mathbf{n} \geq 0, \tag{3.19}$$

where $p \geq 0$ is complementary to $\mathbf{u} \cdot \mathbf{n} \geq 0$. That is, $p > 0$ when $\mathbf{u} \cdot \mathbf{n} = 0$ and $\mathbf{u} \cdot \mathbf{n} > 0$ when $p = 0$.

Figure 3.1: *A selected frame from two simulations of a 3D scenario of fluid splashing inside a spherical boundary. Left: Without separating solid wall boundary conditions, the fluid adheres to the top of the sphere. Right: With separating solid wall boundary conditions, the fluid separates naturally.*

The complementarity condition (3.19) ensures that either the fluid is moving away from the wall, or there is a positive pressure force acting to prevent it from entering the wall. Figure 3.1 (Right) shows that separating solid wall conditions allow the fluid near the top right to separate from the sphere glass.

3.4 Simulation process

Now we discuss how to simulate fluids. Since the analytic solution to the incompressible Euler equations (3.16) is usually difficult to compute, finding numerical solutions is desirable in computer graphics community. As the first equation in (3.16) is nonlinear and the two equations in (3.16) are coupled via the velocity, it is still not easy to numerically solve them as a whole. The splitting method is commonly used in solving Navier-Stokes or Euler equations in fluid simulation [11]. More precisely, the equations (3.16) are solved through splitting into three simpler sub-problems:

$$\frac{D}{Dt}\mathbf{u} = \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{u} = 0. \tag{3.20}$$

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{g}, \tag{3.21}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{1}{\rho}\boldsymbol{\nabla} p = 0 \text{ such that } \boldsymbol{\nabla}\cdot\mathbf{u} = 0, \tag{3.22}$$

where $\frac{D}{Dt}$ is the material derivative operator [11]. Equation (3.20) is called an advection equation, which models the motion of the fluid. Equation (3.21) refers to the acceleration from body forces applying to the velocity field. The pressure changes due to the accumulation of new velocities. Therefore, the pressure equation (3.22) is used to solve the new pressure while ensuring incompressibility. The process of solving the pressure equation is usually called pressure projection, which is the most time consuming part in fluid simulation and the main focus of this thesis. The three equations (3.20) to (3.22) are solve sequentially at each time step. We will explain later why solving these equations is numerically equivalent to solving the original incompressible Euler equations (3.16).

Before giving the details of the simulation algorithm, we introduce some notations for clarification. Assume we are interested in simulation from time $t = 0$ to $t = T$. Let t^n be the n-th time step and $\Delta t^n = t^{n+1} - t^n$ be the time step size. At the n-th time step $t = t^n$, we define the solutions from splitting method $\mathbf{u}^n = (u(\mathbf{x}, t^n), v(\mathbf{x}, t^n), w(\mathbf{x}, t^n))$ as the vector of fluid velocity values and scalar $p^n = p(\mathbf{x}, t^n)$ as the pressure. Given $\mathbf{u}^n$, we want to compute $\mathbf{u}^{n+1}$ for the next time through solving equations (3.20) to (3.22). First, we solve the advection equation (3.20) based on $\mathbf{u}^n$ and denote the solution as $\bar{\mathbf{u}}^n$. Applying forward Euler method to the partial derivative $\frac{\partial \mathbf{u}}{\partial t}$, we have

$$\frac{\bar{\mathbf{u}}^n - \mathbf{u}^n}{\Delta t^n} + (\mathbf{u}^n\cdot\boldsymbol{\nabla})\mathbf{u}^n \approx 0, \tag{3.23}$$

which can be solve by a semi-Lagrangian method [11]. Then we solve the equation (3.21) using forward Euler method. Let the solution be $\hat{\mathbf{u}}^n$, we have

$$\frac{\hat{\mathbf{u}}^n - \bar{\mathbf{u}}^n}{\Delta t^n} \approx \mathbf{g}. \tag{3.24}$$

Finally, solving the pressure equation (3.22) gives

$$\frac{\mathbf{u}^{n+1} - \hat{\mathbf{u}}^n}{\Delta t^n} + \frac{1}{\rho}\boldsymbol{\nabla} p^n \approx 0 \text{ such that } \boldsymbol{\nabla} \cdot \mathbf{u}^{n+1} = 0. \tag{3.25}$$

Adding equations (3.23) to (3.25) together, we have

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t^n} + (\mathbf{u}^n \cdot \boldsymbol{\nabla})\mathbf{u}^n + \frac{1}{\rho}\boldsymbol{\nabla} p^n \approx \mathbf{g} \text{ such that } \boldsymbol{\nabla} \cdot \mathbf{u}^{n+1} = 0. \tag{3.26}$$

This means $\mathbf{u}^{n+1}$, which is computed via splitting method, approximates the solution to the incompressible Euler equations (3.16) at the time step $t = t^{n+1}$. Therefore, the splitting method gives an approximate solution to the incompressible Euler equations. The basic algorithm for fluid simulation is given as follows:

Algorithm 1 Fluid simulation algorithm for numerically solving equations (3.16)

1: Given an initial velocity field $\mathbf{u}^0$.
2: **for** $n = 0, 1, 2, ...$ until $t^n = T$ **do**
3: Determine a suitable time step size Δt^n.
4: Compute $\bar{\mathbf{u}}^n$ through advecting velocity field $\mathbf{u}^n$.
5: Compute $\hat{\mathbf{u}}^n$ by applying body forces using forward Euler method.
6: Compute the pressure p^n and use it to update the new velocity field $\mathbf{u}^{n+1}$ for the next time step.
7: Advance to the next time step $t^{n+1} = t^n + \Delta t^n$.
8: **end for**

To avoid extra numerical dissipation and reduced accuracy caused by an aggressive time step size, it is suggested [32] that the time step size should satisfy the following condition:

$$\Delta t^n \leq \frac{5\Delta x}{u_{max}}, \tag{3.27}$$

where Δx is the space step size, u_{max} is the estimated maximum absolute value of fluid velocity at time t^n.

Now we discuss how to solve (3.22). After solving the equation (3.21), we have the updated solution $\hat{\mathbf{u}}^n$ at time step t^n. We now describe how to use it to compute $\mathbf{u}^{n+1}$ via solving the pressure equation (3.22). First, we discuss the standard pressure projection in fluid simulation, which makes the fluid incompressible with the standard boundary conditions (3.18). Applying forward Euler method to the first equation of (3.22) at the n-th time step gives

$$\mathbf{u}^{n+1} = \hat{\mathbf{u}}^n - \Delta t^n \frac{1}{\rho} \boldsymbol{\nabla} p^n. \tag{3.28}$$

The updated velocity field $\mathbf{u}^{n+1}$ needs to satisfy the incompressibility inside the fluid

$$\boldsymbol{\nabla} \cdot \mathbf{u}^{n+1} = 0. \tag{3.29}$$

Substituting (3.28) into (3.29) gives a Poisson equation for the pressure p^n:

$$\boldsymbol{\nabla} \cdot \mathbf{u}^{n+1} = -\frac{\Delta t^n}{\rho} \boldsymbol{\nabla}^2 p^n + \boldsymbol{\nabla} \cdot \hat{\mathbf{u}}^n = 0. \tag{3.30}$$

It can be shown that the solid wall boundary condition $\mathbf{u}^{n+1} \cdot \mathbf{n} \geq 0$ gives us $\boldsymbol{\nabla} \cdot \mathbf{u}^{n+1} \geq 0$. After enforcing the separating solid wall conditions (3.19) in (3.30), the new equation for p^n satisfying the separating solid wall boundary conditions becomes:

$$0 \leq p^n \perp -\frac{\Delta t^n}{\rho} \boldsymbol{\nabla}^2 p^n + \boldsymbol{\nabla} \cdot \hat{\mathbf{u}}^n \geq 0. \tag{3.31}$$

Derivation details can be found in the work of Andersen et al. [3].

We will discuss in detail how to solve the equations (3.30) and (3.31) numerically in section 3.7. Once p^n is solved, the new velocity $\mathbf{u}^{n+1}$ can be calculated using the formula in (3.28).

3.5 Staggered grid

So far we have presented the fluid simulation process and showed the equations for the pressure at each time step. Now we look into these equations in terms of space and solve them numerically. Before moving into the numerical computation, we discuss the staggered grid or Marker-and-Cell (MAC) grid [38] for spatial discretization. The unique feature of staggered grid is that different variables are stored at different positions; see Figure 3.2 for an illustration in 2D. The black dots represent the locations of pressure while the red

and blue dashes represent the locations of velocities in horizontal and vertical directions, respectively. We denote the pressure at the center of the (i, j)-th cell as $p_{i,j}$. For the velocity $\mathbf{u} = (u, v)$, instead of treating $u_{i,j}$ and $v_{i,j}$ at the center as unknowns, we denote $u_{i+\frac{1}{2},j}$, as the velocity at the center of shared edge of the the (i, j)-th and the $(i+1, j)$-th cells, and $v_{i,j+\frac{1}{2}}$, as the velocity at the center of shared edge of the the (i, j)-th and the $(i, j+1)$-th cells. Therefore, for each cell, we have the pressure at the center, two horizontal velocities at the center of left and right edges (red dash), and two vertical velocities at the center of top and bottom edges (blue dash).

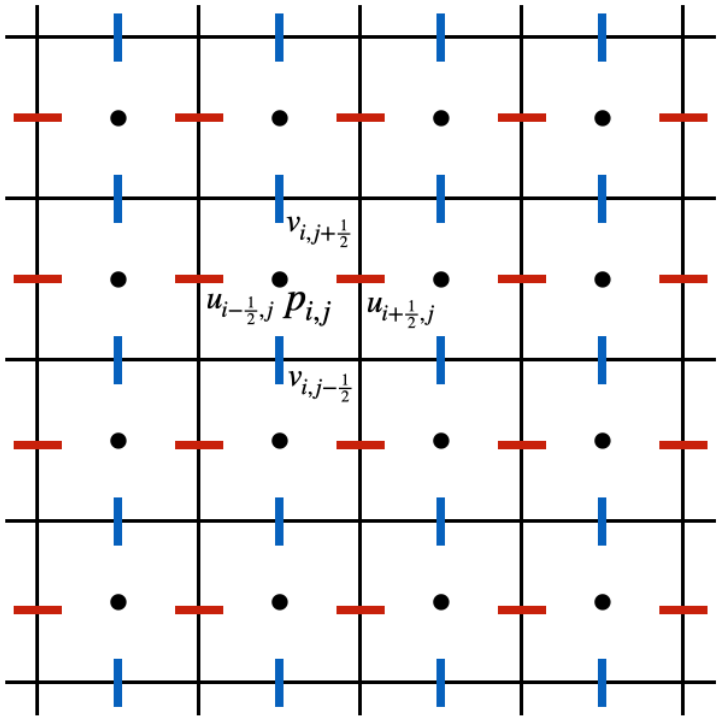


Figure 3.2: *Staggered grid in 2D with pressure stored at the cell centers (black dots) and velocities stored at the centers of the edge (red and blue dashes).*

In 3D, where the grid cell is a cube, the pressure is stored at the center and velocities are stored at the center of the six faces. For the (i, j, k)-th cell, we denote the pressure at the center as $p_{i,j,k}$, the velocities at x direction as $u_{i+\frac{1}{2},j,k}$, $u_{i-\frac{1}{2},j,k}$, y direction as $v_{i,j+\frac{1}{2},k}$, $v_{i,j-\frac{1}{2},k}$, and z direction as $w_{i,j,k+\frac{1}{2}}$, $w_{i,j,k-\frac{1}{2}}$.

3.6 Level set

In fluid simulation, we need to determine which cells fall inside fluid, solid, or air and which cells are partially covered by the fluid. The level set method [51] is commonly used. A level set function $\Phi : \mathbb{R}^d \mapsto \mathbb{R}$ (d is the dimension of the space) is defined as follows:

$$\Phi(\mathbf{x}) = \begin{cases} = 0, \text{ when } \mathbf{x} \text{ is on the surface,} \\ > 0, \text{ when } \mathbf{x} \text{ is outside the fluid,} \\ < 0, \text{ when } \mathbf{x} \text{ is inside the fluid.} \end{cases} \tag{3.32}$$

Given a position $\mathbf{x}$, we compute the minimum distance to the surface and determine the sign of $\Phi(\mathbf{x})$ based on (3.32). For example, if we have a fluid sphere in 3D centered at $(1, 1, 1)$ with radius of 2: $(x-1)^2 + (y-1)^2 + (z-1)^2 = 2^2$, the level set function can be defined as:

$$\Phi(x, y, z) = \sqrt{(x-1)^2 + (y-1)^2 + (z-1)^2} - 2. \tag{3.33}$$

With the level set function, we can use it to calculate the pressure near the surface. For example, for a cell falling into both fluid and air but with its center in the air, we treat it differently instead of setting the pressure to zero. Let's assume that the (i, j, k)-th cell covers only fluid while the $(i+1, j, k)$-th cell falls in both fluid and air, as shown in Figure 3.3. In this case, we have $\Phi(\mathbf{x}_{i,j,k}) < 0$ and $\Phi(\mathbf{x}_{i+1,j,k}) > 0$. The fluid-air surface is crossing the $(i+1, j, k)$-th cell. The pressure $\mathbf{x}_\Gamma$ is set to zero due to the boundary condition. Using the level set function, we can calculate θ as:

$$\theta = \frac{-\Phi(\mathbf{x}_{i,j,k})}{\Phi(\mathbf{x}_{i+1,j,k}) - \Phi(\mathbf{x}_{i,j,k})}, \tag{3.34}$$

and the pressure at the $(i+1, j, k)$-th cell can be calculated as:

$$p_{i+1,j,k} = (1 - \frac{1}{\theta}) p_{i,j,k}, \tag{3.35}$$

which will be used for the discretization at the cell (i, j, k).

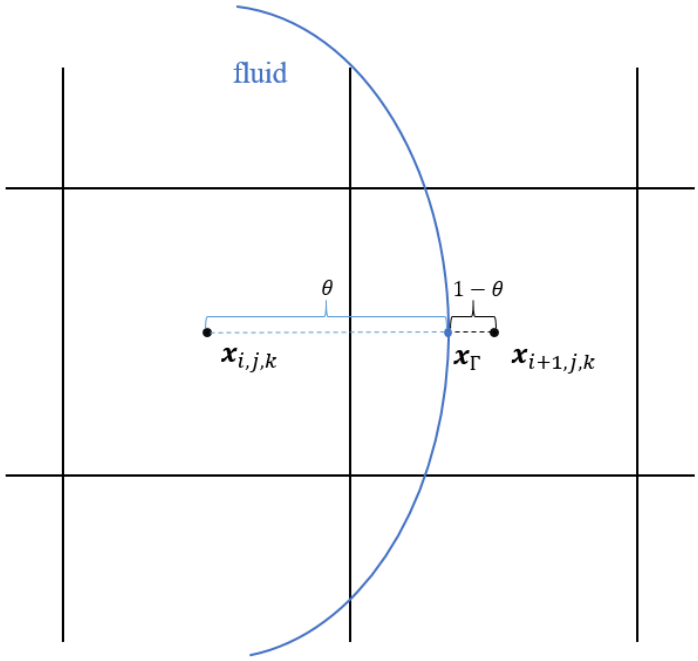


Figure 3.3: *3D projection on the (x, y) plane showing that the cell $(i + 1, j, k)$ partially falls into the fluid*

3.7 Discretization

Now we discuss how to solve the pressure numerically. We first describe the discretization of the pressure equation in 3D assuming the standard boundary conditions are used. The discretization in 2D is similar. Assume that the (i, j, k)-th cell whose center falls into the fluid. Applying central difference method in the 3D staggered grid, (3.30) is discretized as:

$$\begin{aligned}&\frac{\Delta t^n}{\rho}\left(\frac{6p^n_{i,j,k} - p^n_{i+1,j,k} - p^n_{i-1,j,k} - p^n_{i,j+1,k} - p^n_{i,j-1,k} - p^n_{i,j,k+1} - p^n_{i,j,k-1}}{h^2}\right)\\ &= -\left(\frac{\hat{u}^n_{i+\frac{1}{2},j,k} - \hat{u}^n_{i-\frac{1}{2},j,k}}{h} + \frac{\hat{v}^n_{i,j+\frac{1}{2},k} - \hat{v}^n_{i,j-\frac{1}{2},k}}{h} + \frac{\hat{w}^n_{i,j,k+\frac{1}{2}} - \hat{w}^n_{i,j,k-\frac{1}{2}}}{h}\right),\end{aligned} \tag{3.36}$$

where we assume that the grid size in all directions are equal, namely, $\Delta x = \Delta y = \Delta z = h$. If the (i, j, k)-th cell's center does not fall into the fluid, we know that the pressure is zero. Therefore, we have

$$p_{i,j,k} = 0 \tag{3.37}$$

for those cells.

We note that the discretized pressure equation (3.36) needs to be modified when the cell (i, j, k) is near the boundary. For example, if the $(i + 1, j, k)$-th cell falls into the air (as shown in Figure 3.3), we replace the pressure $p^n_{i+1,j,k}$ with $(1 - \frac{1}{\theta})p^n_{i,j,k}$ based on (3.35). Therefore, (3.36) should be modified as follows:

$$\begin{aligned}&\frac{\Delta t^n}{\rho}\Big(\frac{(5+\frac{1}{\theta})p^n_{i,j,k} - p^n_{i-1,j,k} - p^n_{i,j+1,k} - p^n_{i,j-1,k} - p^n_{i,j,k+1} - p^n_{i,j,k-1}}{h^2}\Big)\\ &= -\Big(\frac{\hat{u}^n_{i+\frac{1}{2},j,k} - \hat{u}^n_{i-\frac{1}{2},j,k}}{h} + \frac{\hat{v}^n_{i,j+\frac{1}{2},k} - \hat{v}^n_{i,j-\frac{1}{2},k}}{h} + \frac{\hat{w}^n_{i,j,k+\frac{1}{2}} - \hat{w}^n_{i,j,k-\frac{1}{2}}}{h}\Big).\end{aligned} \tag{3.38}$$

Note that since θ is between 0 and 1, the coefficient of $p_{i,j,k}$ contributed by the cell $(i+1, j, k)$ will be $\frac{1}{\theta} > 1$, which will make the matrix more diagonally dominant. Thus, as long as there is air space inside the solid container, there is always at least one strictly diagonal dominant row in the matrix.

Now we look into the solid-fluid surface. Substituting (3.28) into the boundary condition (3.18), we have

$$\Delta t^n \frac{1}{\rho}\boldsymbol{\nabla} p^n \cdot \mathbf{n} = \hat{\mathbf{u}}^n \cdot \mathbf{n} - \mathbf{u}_{solid} \cdot \mathbf{n}. \tag{3.39}$$

By introducing the fluid volume $V \in [0, 1]$ to deal with cells near the solid, (3.36) becomes [11]:

$$
\begin{aligned}
&\frac{\Delta t^n}{\rho}\Big(\frac{V_{i+\frac{1}{2},j,k}+V_{i-\frac{1}{2},j,k}+V_{i,j+\frac{1}{2},j,k}+V_{i,j-\frac{1}{2},k}+V_{i,j,k+\frac{1}{2}}+V_{i,j,k-\frac{1}{2}}}{h^2}p^n_{i,j,k}\\
&-\frac{V_{i+\frac{1}{2},j,k}}{h^2}p^n_{i+1,j,k}-\frac{V_{i-\frac{1}{2},j,k}}{h^2}p^n_{i-1,j,k}-\frac{V_{i,j+\frac{1}{2},k}}{h^2}p^n_{i,j+1,k}\\
&-\frac{V_{i,j-\frac{1}{2},k}}{h^2}p^n_{i,j-1,k}-\frac{V_{i,j,k+\frac{1}{2}}}{h^2}p^n_{i,j,k+1}-\frac{V_{i,j,k-\frac{1}{2}}}{h^2}p^n_{i,j,k-1}\Big)\\
&=-\Big(\frac{V_{i+\frac{1}{2},j,k}\hat{u}^n_{i+\frac{1}{2},j,k}-V_{i-\frac{1}{2},j,k}\hat{u}^n_{i-\frac{1}{2},j,k}}{h}+\frac{V_{i,j+\frac{1}{2},k}\hat{v}^n_{i,j+\frac{1}{2},k}-V_{i,j-\frac{1}{2},k}\hat{v}^n_{i,j-\frac{1}{2},k}}{h}\\
&+\frac{V_{i,j,k+\frac{1}{2}}\hat{w}^n_{i,j,k+\frac{1}{2}}-V_{i,j,k-\frac{1}{2}}\hat{w}^n_{i,j,k-\frac{1}{2}}}{h}\Big)\\
&+\frac{V_{i+\frac{1}{2},j,k}-V_{i,j,k}}{h}\hat{u}^{solid}_{i+\frac{1}{2},j,k}-\frac{V_{i-\frac{1}{2},j,k}-V_{i,j,k}}{h}\hat{u}^{solid}_{i-\frac{1}{2},j,k}\\
&+\frac{V_{i,j+\frac{1}{2},k}-V_{i,j,k}}{h}\hat{u}^{solid}_{i,j+\frac{1}{2},k}-\frac{V_{i,j-\frac{1}{2},k}-V_{i,j,k}}{h}\hat{u}^{solid}_{i,j-\frac{1}{2},k}\\
&+\frac{V_{i,j,k+\frac{1}{2}}-V_{i,j,k}}{h}\hat{u}^{solid}_{i,j,k+\frac{1}{2}}-\frac{V_{i,j,k-\frac{1}{2}}-V_{i,j,k}}{h}\hat{u}^{solid}_{i,j,k-\frac{1}{2}},
\end{aligned}
\tag{3.40}
$$

where $\hat{u}^{solid}$, $\hat{v}^{solid}$, $\hat{w}^{solid}$ are the velocities of solid objects along x, y, z axes, respectively.

Putting (3.36), (3.37), (3.38), and (3.40) together, we obtain a linear system as follows:

$$\mathbf{Ap}+\mathbf{b}=\mathbf{0}, \tag{3.41}$$

where the matrix $\mathbf{A}$ corresponds to the discretization of the operator $-\frac{\Delta t}{\rho}\boldsymbol{\nabla}^2$, $\mathbf{b}$ and $\mathbf{p}$ are vectors for values of $\boldsymbol{\nabla}\cdot\hat{\mathbf{u}}^n$ and pressure p^n, respectively. Each of their entries corresponds to the value at the center of a grid cell.

Based on the discretization, the matrix $\mathbf{A}$ is an M-Matrix [13] (see the Appendix A for detailed proof) and symmetric positive definite (SPD) [12]. Since (3.41) is a linear system with a SPD matrix, the preconditioned conjugated gradient method (PCG) is usually used for solving the pressure in fluid simulation.

If the separating solid boundary conditions are used, the discretization of the pressure p^n gives a LCP problem (which will be discussed in detail in Chapter 4) as follows:

$$0\leq\mathbf{p}\perp\mathbf{Ap}+\mathbf{b}\geq 0. \tag{3.42}$$

After solving either the linear system (3.41) or the LCP (3.42) for p^n, we can use central difference method again in (3.28) to update the velocity field in each direction as follows:

$$u^{n+1}_{i+\frac{1}{2},j,k} = \hat{u}^{n}_{i+\frac{1}{2},j,k} - \frac{\Delta t}{\rho}\frac{p^{n}_{i+1,j,k} - p^{n}_{i,j,k}}{h}, \tag{3.43}$$

$$v^{n+1}_{i,j+\frac{1}{2},k} = \hat{v}^{n}_{i,j+\frac{1}{2},k} - \frac{\Delta t}{\rho}\frac{p^{n}_{i,j+1,k} - p^{n}_{i,j,k}}{h}, \tag{3.44}$$

$$w^{n+1}_{i,j,k+\frac{1}{2}} = \hat{w}^{n}_{i,j,k+\frac{1}{2}} - \frac{\Delta t}{\rho}\frac{p^{n}_{i,j,k+1} - p^{n}_{i,j,k}}{h}. \tag{3.45}$$

Chapter 4

LCP formulation from pressure equation

In the previous chapter, we briefly mentioned that the discretization of the pressure gives a linear complementarity problem (LCP) when separating solid boundary conditions are used. In this chapter, we first describe the definition of the LCP and its applications and then discuss in detail how to formulate the LCP from the pressure equation with seprarating solid boundary conditions.

4.1 Linear complementarity problem (LCP)

We define the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and the column vector $\mathbf{b} \in \mathbb{R}^{n \times 1}$. Assume that $\mathbf{A}$ and $\mathbf{b}$ are constants, we aim to find the vector $\mathbf{x} \in \mathbb{R}^{n \times 1}$ such that, for all $1 \leq i \leq n$,

$$\mathbf{x}_i \geq 0, \tag{4.1}$$

$$(\mathbf{A}\mathbf{x} + \mathbf{b})_i \geq 0, \tag{4.2}$$

$$\mathbf{x}_i(\mathbf{A}\mathbf{x} + \mathbf{b})_i = 0, \tag{4.3}$$

where $\mathbf{x}_i$ is the i-th entry of $\mathbf{x}$ and $(\mathbf{A}\mathbf{x} + \mathbf{b})_i$ is the i-th entry of $\mathbf{A}\mathbf{x} + \mathbf{b}$. The above conditions (4.1) to (4.3) are known as linear complementarity conditions and are usually expressed as follows:

$$0 \leq \mathbf{x} \perp \mathbf{A}\mathbf{x} + \mathbf{b} \geq 0. \tag{4.4}$$

Here $\perp$ means the condition on the left is complementary to the condition on the right. In other words, if $\mathbf{x}_i > 0$ we have $(\mathbf{Ax} + \mathbf{b})_i = 0$ or if $(\mathbf{Ax} + \mathbf{b})_i > 0$ we have $\mathbf{x}_i = 0$. The LCP conditions (4.1) to (4.3) can be viewed from different perspectives.

For example, the LCP can be formulated as root finding problem and solved with Newton's method [25]. Let $\mathbf{y} = \mathbf{Ax} + \mathbf{b}$ and define $\min(a, b)$ as the minimum value of a and b. Let $\mathbf{y}_i$ be the i-th entry of $\mathbf{y}$. Since $\min(\mathbf{x}_i, \mathbf{y}_i) = 0$ means that either $\mathbf{x}_i$ or $\mathbf{y}_i$ has to be 0, we have $\mathbf{y}_i = 0$ if $\mathbf{x}_i > 0$ or $\mathbf{x}_i = 0$ if $\mathbf{y}_i > 0$, which is the definition of the LCP conditions. Therefore, the LCP problem (4.4) is equivalent to solving the following problem:

$$\mathbf{\Omega}(\mathbf{x}) = \begin{pmatrix} \min(\mathbf{x}_1, \mathbf{y}_1) \\ ... \\ \min(\mathbf{x}_n, \mathbf{y}_n) \end{pmatrix} = \mathbf{0}. \tag{4.5}$$

Besides the minimum function ω, another popular function used for reformulating LCP is the Fischer-Burmeister function defined as follows:

$$\phi(x, y) \equiv \sqrt{x^2 + y^2} - x - y. \tag{4.6}$$

When $\phi(x, y) = 0$, it is easy to obtain that $xy = 0$, which means either $x = 0$ or $y = 0$. If $x = 0$, only $y \geq 0$ can satisfy $\phi(x, y) = 0$. Similarly, we have $x \geq 0$ when $y = 0$. This means both x and y are non-negative and if either is positive, the other must be zero. Therefore, $\phi(x, y) = 0$ is equivalent to $0 \leq x \perp y \geq 0$. The LCP problem (4.4) can be converted to:

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} \phi(\mathbf{x}_1, \mathbf{y}_1) \\ ... \\ \phi(\mathbf{x}_n, \mathbf{y}_n) \end{pmatrix} = \mathbf{0}. \tag{4.7}$$

The LCP can also be reformulated by introducing a control, which gives a Hamilton-Jacobi-Bellman (HJB) Equation. As we know from (4.5), for the i-th entry ($1 \leq i \leq n$), we have

$$\min \left(\mathbf{x}_i, \sum_{j=1}^{n} \mathbf{A}_{i,j} \mathbf{x}_j + \mathbf{b}_i \right) = 0. \tag{4.8}$$

Introducing the control variable λ_i whose value is either 0 or 1, we rewrite the above equation as follows:

$$\min_{\lambda_i \in \{0,1\}} \left\{ \lambda_i \left(\sum_{j=1}^{n} \mathbf{A}_{i,j} \mathbf{x}_j + \mathbf{b}_i \right) + (1 - \lambda_i) \mathbf{x}_i \right\} = 0, \tag{4.9}$$

which is equivalent to the conditions (4.1) to (4.3). To write (4.9) as the matrix form, we define the set $\mathscr{M}_n = \{diag(\{\lambda_i\})|\lambda_i = 0 \text{ or } 1\}$. We use the matrix $\Lambda = diag(\{\lambda_i\})$ as the control, where $diag(\{\lambda_i\})$ is a square diagonal matrix with the elements of vector $\{\lambda_i\}$ on the main diagonal. Therefore, (4.9) can be written as:

$$\inf_{\Lambda \in \mathscr{M}_n} \{\Lambda(\mathbf{A}\mathbf{x} + \mathbf{b}) + (\mathbf{I} - \Lambda)\mathbf{x}\} = 0, \tag{4.10}$$

where $\mathbf{I}$ is the identity matrix.

The LCP problem is also equivalent to the quadratic optimization as follows:

$$\min \left(\frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} + \mathbf{b}^T \mathbf{x} \right) \quad s.t. \quad \mathbf{x} \geq \mathbf{0}. \tag{4.11}$$

The detailed proof can be found in [9].

The LCP problems arise from different applications. For example, in American option pricing, computing the fair price with underlying asset following a jump diffusion process can be formulated as a LCP problem [20]. Besides the fluid simulation, LCP also can be found in other areas (rigid body for example) in computer graphics [61, 53, 55].

4.2 LCP formulation

Now we discuss how to formulate LCP from the discretization of the pressure with separating solid boundary conditions. At first, we revisit the staggered grid in 2D and illustrate how the pressure is stored in different types of cells. Let's assume we have a circular domain in 2D with fluid filling its right half space, as shown in Figure 4.1. We assume there are N_C grid cells in the discretized domain and index them from 1 to N_C. Specifically, the cell (i_x, i_y) will be denoted as the i-th grid cell where $i = i_x + (i_y - 1)N_y$ and N_y is the number of grid cells along the y axis.

To prevent the fluid from sticking to the solid wall, the separating solid boundary conditions (3.19) only need to be applied to boundary points. Let $\mathbf{p}_i$ be the pressure of the i-th grid cell in the discretization domain. If i corresponds to a cell near the solid boundary (grey), the separating wall boundary conditions are enforced and the i-th row of the discretized pressure equation (3.42) is

$$0 \leq \mathbf{p}_i \perp \sum_{j=1}^{N_C} \mathbf{A}_{i,j}\mathbf{p}_j + \mathbf{b}_i \geq 0, \tag{4.12}$$

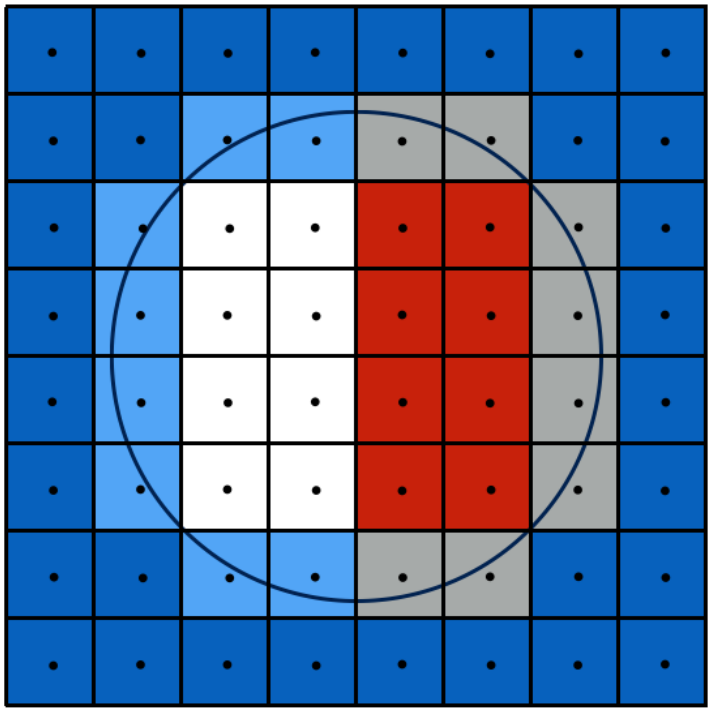


Figure 4.1: *Grid cells after discretization in 2D for a circular domain whose right half contains fluid. Red: cells in the fluid; Blue: cells in the solid; White: cells in the air; Grey: cells in both fluid and solid; Azure: cells in both solid and air.*

which can also be written as an HJB equation

$$\min_{\lambda_i \in \{0,1\}} \{\lambda_i (\sum_{j=1}^{N_C} \mathbf{A}_{i,j}\mathbf{p}_j + \mathbf{b}_i) + (1-\lambda_i)\mathbf{p}_i\} = 0, \tag{4.13}$$

where λ_i is the control that minimizes the term in the braces on the left hand side. The solution $\mathbf{p}_i$ and the control λ_i are unknown and depend on each other. If the i-th cell completely falls outside of the fluid (white, azure, or blue) or inside the fluid (red), the i-th row of the discretized pressure equation simply becomes linear:

$$\sum_{j=1}^{N_C} \mathbf{A}_{i,j}\mathbf{p}_j + \mathbf{b}_i = 0. \tag{4.14}$$

Defining $\mathscr{S} = \{1 \le i \le N_C |$ The i-th cell lies in both fluid and solid$\}$ (grey cells in Figure 4.1), then the LCP problem (3.42) can be converted into the following mixed LCP (or

MLCP):

$$\begin{cases} \min_{\lambda_i \in \{0,1\}} \{\lambda_i (\sum_{j=1}^{N_C} \mathbf{A}_{i,j} \mathbf{p}_j + \mathbf{b}_i) + (1 - \lambda_i) \mathbf{p}_i\} = 0, \text{ if } i \in \mathscr{S} \\ \sum_{j=1}^{N_C} \mathbf{A}_{i,j} \mathbf{p}_j + \mathbf{b}_i = 0, \text{ otherwise.} \end{cases} \tag{4.15}$$

We define the set of possible control matrices as $\mathscr{M} = \{diag(\{\lambda_i\}) | \lambda_i = 0 \text{ or } 1 \text{ if } i \in \mathscr{S}; \lambda_i = 1 \text{ if } i \notin \mathscr{S}\}$. The nonlinear equation (4.15) can be written as an HJB equation:

$$\inf_{\Lambda \in \mathscr{M}} \{\Lambda(\mathbf{A}\mathbf{p} + \mathbf{b}) + (\mathbf{I} - \Lambda)\mathbf{p}\} = 0, \tag{4.16}$$

where Λ serves as the control and inf is performed componentwise on the vector inside the braces. We will discuss how to solve this equation in Chapter 5.

Chapter 5

Fast solvers

The LCP problem is a nonlinear problem, and therefore efficient solvers for linear systems cannot be used directly. Solving the LCP equation (4.15) is challenging as the solution and the constraints are coupled with each other. It requires enforcing the constraints while the solution must be known in order to know where to enforce the constraints.

Several approaches have been proposed to solve LCP problems. However, they either have slow convergence or do not scale well for large problems. Nonlinear iterative methods are used to solve the LCP problems but in general convergence is not guaranteed [13]. Direct approaches like pivoting method suffer from exponential computational cost [25]. The most recent methods, the minimum map Newton method by Andersen et al. [3, 25] and the full multigrid by Chentanez & Muller [14], demonstrated improvement on previous works. However, the Newton method still suffers from scalability issues since it relies on the CG solver, which is not scalable. The full multigrid is based on the standard multigrid for linear system and its scalability has not been explored yet.

To solve the LCP problem in fluid simulation more efficiently, we propose three methods: policy iteration [30], penalty method [31], and full approximation scheme (FAS) multigrid [37], which is a multigrid method for nonlinear equations. We choose policy iteration and penalty method because they converge fast in general. To deal with LCPs arising in larger scale fluid simulations, we prefer a multigrid method due to its scalability; that is, its iteration count is expected to be essentially independent of grid resolution. We will show that our proposed methods can outperform the naive multigrid approach proposed by Chentanez et al. [14] and the non-smooth Newton method proposed by Andersen et al. [3].

5.1 Policy iteration

The policy iteration, also referred as Howard's algorithm [39], is used to solve nonlinear problems like the Bellman's equation from the Markov Decision process (MDP). A typical nonlinear problem that can be solved by policy iteration is given as follows:

$$\min_{\pi}\{\mathbf{M}(\pi)\mathbf{x}+\mathbf{D}(\pi)\}=0, \tag{5.1}$$

where $\mathbf{M}(\pi)$ and $\mathbf{D}(\pi)$ are the matrix and the vector depending on the policy π, respectively. $\mathbf{x}$ is the solution such that the minimum value of $\mathbf{M}(\pi)\mathbf{x}+\mathbf{D}(\pi)$ is zero. The goal is to find an optimal policy π^* and a solution to satisfy (5.1). We note that both $\mathbf{x}$ and π are unknown. The basic idea of policy iteration is to solve one of the unknowns by fixing the other, and iteratively repeat the process until convergence. Therefore, the nonlinear problem (5.1) is solved by alternating between two steps:

- Policy improvement:
$$\pi=\text{argmin}_{\pi}\{\mathbf{M}(\pi)\mathbf{x}+\mathbf{D}(\pi)\}. \tag{5.2}$$
- Policy evaluation:
$$\mathbf{M}(\pi)\mathbf{x}+\mathbf{D}(\pi)=0. \tag{5.3}$$

We first evaluate the optimal policy based on the approximate solution, linearize the problem with the evaluated policy, and then solve the linear system to obtain a better approximate solution. Specifically, given an approximate solution $\mathbf{x}^k$, we compute the improved policy π^k using (5.2) by letting $\mathbf{x}=\mathbf{x}^k$. Fixing the policy $\pi=\pi^k$ to linearize the problem, we solve the linear system (5.3) for the improved approximation $\mathbf{x}^{k+1}$. We repeat this procedure from $k=0$ until convergence.

Compared with Newton method, the linearization process is simpler and less expensive. We solve the LCP (4.16) using the policy iteration by treating the control Λ as the policy π and writing the term inside the braces as $\mathbf{M}(\Lambda)\mathbf{x}+\mathbf{D}(\Lambda)$ where $\mathbf{M}(\Lambda)=\Lambda\mathbf{A}+\mathbf{I}-\Lambda$ and $\mathbf{D}(\Lambda)=\Lambda\mathbf{b}$. Our proposed policy iteration scheme for solving (4.16) first computes the control Λ^k using an initial guess of $\mathbf{p}^k$, and uses the control to linearize the problem. The optimal control is calculated by enumerating all possible values of λ_i (either 0 or 1) for each entry $i\in\mathscr{S}$ on the diagonal of the control matrix and choosing the value such that $\lambda_i(\sum_{j=1}^{N_C}\mathbf{A}_{i,j}\mathbf{p}_j+\mathbf{b}_i)+(1-\lambda_i)\mathbf{p}_i$ is minimum. With the control Λ^k, we can linearize the LCP problem as follows:

$$\Lambda^k(\mathbf{A}\mathbf{p}+\mathbf{b})+(\mathbf{I}-\Lambda^k)\mathbf{p}=\mathbf{0}, \tag{5.4}$$

which can be rewritten as

$$(\Lambda^k \mathbf{A} + \mathbf{I} - \Lambda^k)\mathbf{p} = -\Lambda^k \mathbf{b}. \tag{5.5}$$

We note that the matrix $\Lambda^k \mathbf{A} + \mathbf{I} - \Lambda^k$ may not be symmetric due to the multiplication by Λ^k on the left. We explain here how to convert it to an equivalent linear system with a symmetric matrix. Since Λ^k is a diagonal matrix with its diagonal entries being either 0 or 1, we have $(\Lambda^k)^2 = \Lambda^k$. We can rewrite it as $(\mathbf{I} - \Lambda^k)\Lambda^k = \mathbf{0}$. If we multiply (5.5) with $\mathbf{I} - \Lambda^k$ on both sides, we obtain $(\mathbf{I} - \Lambda^k)\mathbf{p} = \mathbf{0}$, namely $\mathbf{p} = \Lambda^k \mathbf{p}$. Therefore, we have $\Lambda^k \mathbf{A}\mathbf{p} = \Lambda^k \mathbf{A}\Lambda^k \mathbf{p}$. This means, we can multiply the matrix $\Lambda^k \mathbf{A}$ in (5.5) by Λ^k on the right and give the equivalent equation to (5.5) as follows:

$$(\Lambda^k \mathbf{A}\Lambda^k + \mathbf{I} - \Lambda^k)\mathbf{p} = -\Lambda^k \mathbf{b}. \tag{5.6}$$

We then solve the linear system (5.6) to compute an updated solution $\mathbf{p}^{k+1}$ and use it to once again compute a new control Λ^{k+1}. We repeat this process from $k = 0$ until the residual is sufficiently small. Our policy iteration scheme is presented in Algorithm 2. The control update in (5.7) amounts to explicitly enforcing the LCP constraints, by choosing for each row a λ_i that produces the minimum value. The i-th row becomes trivial if λ_i is chosen as 0. The method is remarkably simple, but we shall see that it is quite effective in practice. We calculate the initial guess for the current time step using the control from the previous time step since this exhibited the better performance than zero initial guess in practice.

Algorithm 2 Policy iteration for solving the LCP problem (4.15)

1: Choose an initial guess for $\mathbf{p}^0$ and a tolerance ϵ.
2: **for** $k = 0, 1, 2, ...$ until residual $< \epsilon$ **do**
3: Find the optimal control matrix Λ^k such that

$$\Lambda^k = \arg \inf_{\Lambda \in \mathscr{M}} \{\Lambda(\mathbf{A}\mathbf{p}^k + \mathbf{b}) + (\mathbf{I} - \Lambda)\mathbf{p}^k\}. \tag{5.7}$$

4: Solve the linear system for $\mathbf{p}^{k+1}$,

$$(\Lambda^k \mathbf{A}\Lambda^k + \mathbf{I} - \Lambda^k)\mathbf{p}^{k+1} = -\Lambda^k \mathbf{b}. \tag{5.8}$$

5: **end for**
6: The approximate solution is given by $\mathbf{p}^{k+1}$ after the residual reaches the tolerance ϵ.

We remark that the matrix $\mathbf{A}$ obtained from the pressure equation is an M-Matrix (see section 3.7 for details). The matrix in (5.8) essentially replaces some rows of $\mathbf{A}$ by putting one on the diagonal since λ_i is either 0 or 1. Such replacement only happens to the indices corresponding to the cells near the fluid-solid boundary and will not affect the connectivity property required as an M-Matrix. Thus the matrix in (5.8) is also an M-matrix. Theorem 6.2 in [30] showed that the M-matrix property ensures the sequence of approximate solution is monotone and hence our policy iteration is guaranteed to converge to a unique solution.

We will consider two possible iterative methods to solve the inner linear system (5.8) in Algorithm 2. One is to use preconditioned conjugate gradient (PCG) [21] with incomplete LU factorization (ILU) as the preconditioner, which has been applied successfully to problems in computational fluid dynamics [41, 29, 42]. We use the SPARSEIT++ pacakge [28] to solve the linear system due to its efficiency and robustness. Although the incomplete Cholesky (IC) preconditioner is also used in fluid problems, we choose ILU only because IC is not provided by the linear solver package. Since the matrix is an M-matrix and symmetric, it is guaranteed to converge. However, the rate of convergence may depend on the mesh size. Another attractive option is to use multigrid which will be described in section 5.3. Multigrid is known for mesh-independent convergence, although special care needs to be taken to capture the irregular geometry and boundary conditions. Note that the multigrid scheme used here is a basic *linear* multigrid method (e.g., [48]), distinct from the nonlinear variant we develop in section 5.3.

5.2 Penalty method

In contrast to policy iteration, our proposed penalty method approach solves the LCP by enforcing the constraints implicitly through large penalties. As a result, formulating the linear system in our penalty method is faster than policy iteration, while the linear system in policy iteration has a smaller size. Considering their tradeoffs, we propose to apply both methods to solve the fluid LCP and explore how they perform.

The penalty method is popular in solving constrained optimization problems. Consider the quadratic optimization (4.11) which is equivalent to our LCP problem. The idea of penalty methods is to use a large positive penalty term ρ to penalize the violation of the constraint $\mathbf{x} \geq 0$. We rewrite the problem (4.11) as unconstrained optimization by adding the penalty term and replacing $\mathbf{x}$ with the pressure $\mathbf{p}$ as follows:

$$\min\left(\frac{1}{2}\mathbf{p}^T\mathbf{A}\mathbf{p}+\mathbf{b}^T\mathbf{p}+\rho\sum_{i=0}^{n-1}\max(-\mathbf{p}_i,0)^2\right)=\min(f(\mathbf{p})), \tag{5.9}$$

where we expect the solution approximates the solution of (5.9) to the LCP. The reason is as follows. Suppose there is a severe violation of the constraint $\mathbf{p}\geq 0$, which means $\mathbf{p}$ has some negative entry $\mathbf{p}_i<0$ with very large absolute value. The penalty term $\rho\max(-\mathbf{p}_i,0)^2$ will become very large, thus minimizing the possibility for $\mathbf{p}$ to be selected as the solution. On the other hand, since the constraints $\mathbf{p}_i\geq 0$ are enforced, minimizing $f(\mathbf{p})$ is equivalent to minimizing the quadratic function $\frac{1}{2}\mathbf{p}^T\mathbf{A}\mathbf{p}+\mathbf{b}^T\mathbf{p}$ for $\mathbf{p}\geq 0$, which is exactly the LCP equivalence as shown in (4.11). Since the function to be minimized, namely, $f(\mathbf{p})$ is convex due to the SPD matrix $\mathbf{A}$, letting its gradient be zero will give us the solution:

$$\boldsymbol{\nabla} f(\mathbf{p})=\mathbf{A}\mathbf{p}+\mathbf{b}-\rho\max(-\mathbf{p},0)=0. \tag{5.10}$$

Assuming that $\mathbf{p}$ satisfies the penalized nonlinear equation (5.10), then $\mathbf{A}\mathbf{p}+\mathbf{b}\geq 0$ always holds. If $\mathbf{p}_i\geq 0$, we have $(\mathbf{A}\mathbf{p}+\mathbf{b})_i=0$. If $\mathbf{p}_i<0$, $\mathbf{p}_i=-\frac{1}{\rho}(\mathbf{A}\mathbf{p}+\mathbf{b})_i$ is expected to be very small due to the large penalty parameter ρ (10^9 for all our experiments). Given an initial guess $\mathbf{p}^k$, the penalty method formulates a linear system by comparing each entry with the constraint. It adds a large penalty to the corresponding diagonal of the matrix $\mathbf{A}$ when there is a constraint violation. Note that the penalty only applies to rows corresponding to cells on the fluid-solid boundary. Let the diagonal penalty matrix $\Pi(\mathbf{p})$ be defined as:

$$\Pi(\mathbf{p})_{i,i}=\begin{cases}1, \text{ if } i\in\mathscr{S} \text{ and } \mathbf{p}_i<0\\ \\ 0, \text{ otherwise.}\end{cases} \tag{5.11}$$

Based on (5.10), we have

$$\mathbf{A}\mathbf{p}+\mathbf{b}+\rho\Pi(\mathbf{p}^k)\mathbf{p}=0. \tag{5.12}$$

Therefore, the linear system formulated through $\mathbf{p}^k$ is given as follows:

$$(\mathbf{A}+\rho\Pi(\mathbf{p}^k))\mathbf{p}=-\mathbf{b}, \tag{5.13}$$

Solving the linear system (5.13) gives a new approximate solution $\mathbf{p}^{k+1}$, which is used to formulate a new linear system. This process is started from $k=0$ until convergence. Our penalty method for the nonlinear problem (4.15) is given by Algorithm 3.

Since $\mathbf{A}$ is an M-matrix as mentioned in section 3.7, the matrix in (5.14), which is constructed by adding some positive values to the diagonals of $\mathbf{A}$, is also an M-matrix.

Therefore, the convergence of penalty iteration to a unique solution is guaranteed [31]. As we did for policy iteration, we also propose to solve the inner linear systems using incomplete LU-preconditioned CG [21] or (linear) multigrid. Similar to policy iteration, the initial guess for the current time step is calculated using the penalty matrix (5.11) from the previous time step to improve the solver's performance.

Algorithm 3 Penalty method for solving the LCP problem (4.15)

1: Choose an initial guess for $\mathbf{p}^0$, a tolerance ϵ, and the discount $\rho \gg \epsilon$.
2: **for** $k = 0, 1, 2, ...$ until residual $< \epsilon$ **do**
3: Solve the linear system for $\mathbf{p}^{k+1}$,

$$(\mathbf{A} + \rho\Pi(\mathbf{p}^k))\mathbf{p}^{k+1} = -\mathbf{b}, \tag{5.14}$$

4: **end for**
5: The approximate solution is given by $\mathbf{p}^{k+1}$ after the residual reaches the tolerance ϵ.

5.3 Multigrid

Both our policy iteration and penalty method have nested iterations: the outer iteration for updating a linear system and the inner iteration for solving the linear system. Since this can become computationally expensive, especially for larger problems, in this section we propose an efficient multigrid method for the LCP problem. While Chentanez et al. [14] previously proposed a multigrid scheme to solve LCPs from fluid simulations, their method is based on standard multigrid for linear problems. It is therefore expected to achieve sub-optimal performance and scaling behavior on our nonlinear problem. Instead, we propose to use the full approximation scheme (FAS) [57, 10, 37], which is a multigrid framework designed specifically for nonlinear problems.

In this section, we will introduce our multigrid method and describe how it can be applied to solve the LCP (4.15). But first, we will explain the idea of multigrid for solving the pressure equation without the LCP condition. Multigrid has two main components: smoothing, which removes high frequency errors on a fine grid to make the error smooth (see the example in Figure 5.1); and coarse grid correction, which removes the low frequency errors on a coarser grid. It proceeds from the fine grid to the coarse grid, solves the equation exactly, and transfers information back to correct the fine grid error. This process is called a V-cycle and can be extended to multiple grid levels by applying it recursively (see Figure 5.2). A standard multigrid method iterates the V-cycle until convergence.

Based on the V-cycle, the more sophisticated scheme to do multigrid is called full multigrid V-cycle (FMG). The idea is to firstly obtain a better initial guess from coarse grid before performing the V-cycle as shown in Figure 5.2. It first starts with solving the equation on the coarsest grid with grid size $8h$ as shown in Figure 5.3. Then it interpolates the results to the next finer grid, namely the grid with grid size $4h$. On this finer grid, it performs a V-cycle and interpolates the results again to the next finer grid with the grid size $2h$. This process is done recursively until the finest grid (of size h for the case in Figure 5.3) is reached.

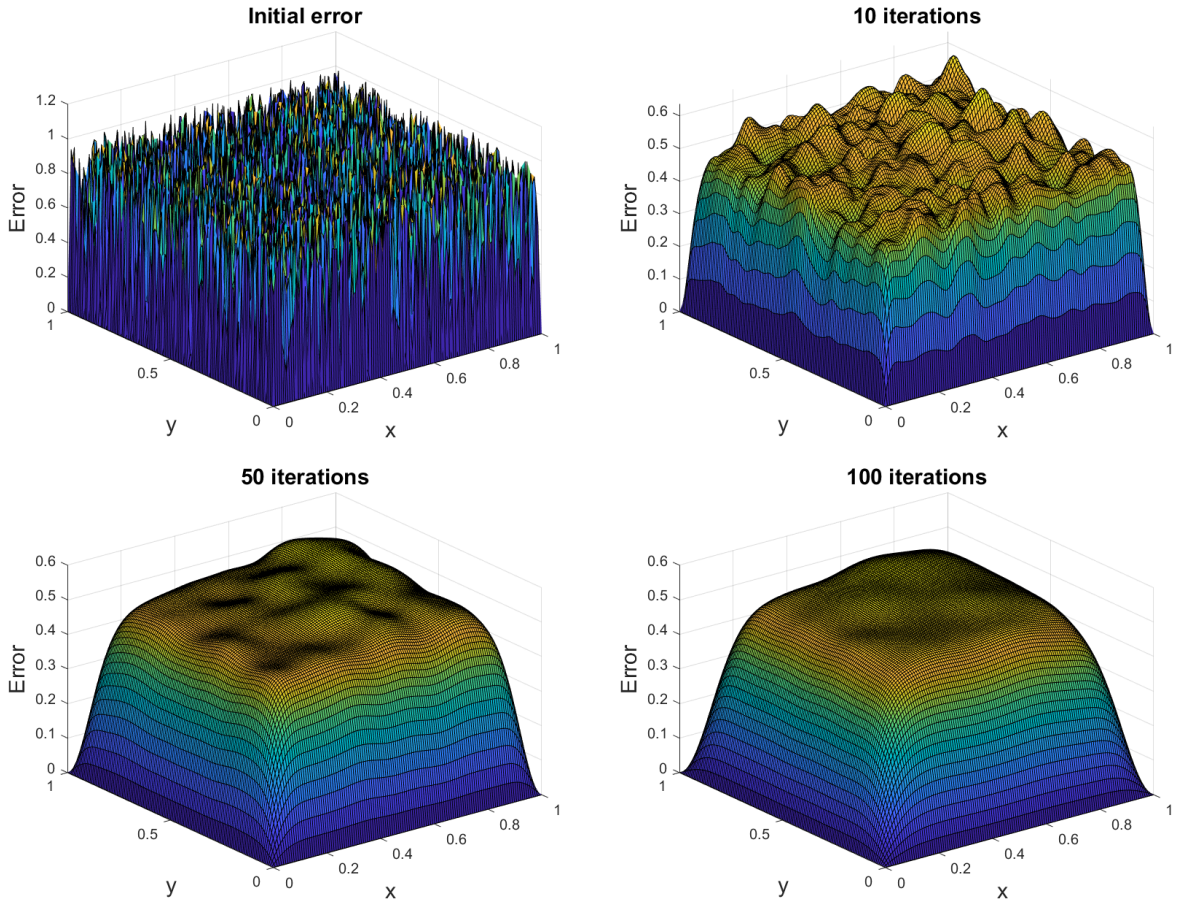


Figure 5.1: *The error plot for a 2D Poisson problem after applying Gauss-Seidel smoother with 0, 10, 50, and 100 iterations. We can see that the error becomes smoother after 10 iterations but is not reduced much from 50 to 100 iterations.*

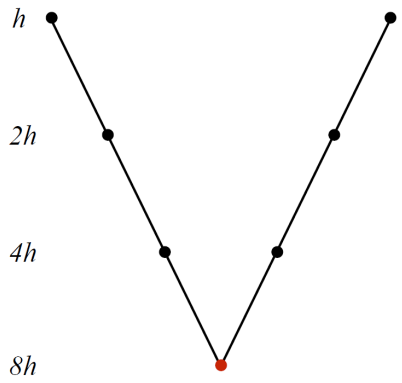


Figure 5.2: *A V-cycle in the 4-level multigrid from the finest grid with grid size h to the coarsest grid with grid size 8h.*

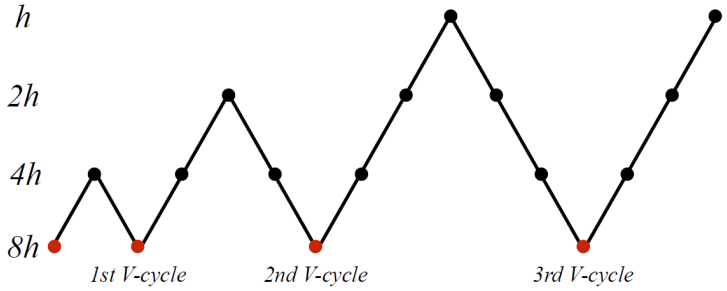


Figure 5.3: *A full-cycle in the 4-level multigrid from the finest grid with grid size h to the coarsest grid with grid size 8h. This full cycle contains 3 V-cycles.*

To illustrate multigrid in detail, consider solving a linear system $\mathbf{A}^h\mathbf{p}^h = \mathbf{f}^h$ discretized from a domain with the grid size h, where $\mathbf{A}^h$ is the discrete Laplacian matrix, $\mathbf{f}^h$ is a column vector and $\mathbf{p}^h$ is the exact solution. Let $\tilde{\mathbf{p}}^h$ be the approximate solution after pre-smoothing (e.g. damped Jacobi, Gauss-Seidel). We define the fine grid error as

$$\mathbf{e}^h = \mathbf{p}^h - \tilde{\mathbf{p}}^h, \tag{5.15}$$

and the corresponding residual as

$$\mathbf{r}^h = \mathbf{f}^h - \mathbf{A}^h\tilde{\mathbf{p}}^h = \mathbf{A}^h(\mathbf{p}^h - \tilde{\mathbf{p}}^h) = \mathbf{A}^h\mathbf{e}^h. \tag{5.16}$$

Since resolving the smoothed error $\mathbf{e}^h$ on the coarse grid is more effective, we restrict the residual equation $\mathbf{A}^h\mathbf{e}^h = \mathbf{r}^h$ to the coarse grid with grid size H and obtain

$$\mathbf{A}^H\mathbf{e}^H = \mathbf{r}^H, \tag{5.17}$$

where $\mathbf{A}^H$ is the matrix constructed by discretizing the PDE on the coarse grid and $\mathbf{r}^H$ is restriction of $\mathbf{r}^h$ from fine to coarse grid. Interpolating the solution $\boldsymbol{\epsilon}^H$ to the coarse grid problem (5.17) gives us an approximate solution to $\mathbf{A}^h\mathbf{e}^h = \mathbf{r}^h$. We use it to correct the approximate solution on the fine grid $\tilde{\mathbf{p}}^h$ as $\bar{\mathbf{p}}^h \leftarrow \tilde{\mathbf{p}}^h + P(\boldsymbol{\epsilon}^H)$. Post-smoothing is applied to $\bar{\mathbf{p}}^h$ after the correction. The multigrid method iterates the V-cycle until convergence. The details of a V-cycle for linear multigrid is given in Algorithm 4.

Algorithm 4 V-Cycle of the multigrid for solving the linear system (3.41)

$\hat{\mathbf{p}}^h_{new} \leftarrow$ V-Cycle$(\mathbf{A}^h, \mathbf{f}^h, \hat{\mathbf{p}}^h)$:
Define the restriction operator R and interpolation operator P.
if h is the coarsest level **then**
    Solve $\mathbf{A}^h\mathbf{p}^h = \mathbf{f}^h$.
else
    Pre-smooth with appropriate smoother:
    $\tilde{\mathbf{p}}^h = presmoothing(\mathbf{A}^h, \mathbf{f}^h, \hat{\mathbf{p}}^h)$.
    Compute the residual $\mathbf{r}^h = \mathbf{f}^h - \mathbf{A}^h\tilde{\mathbf{p}}^h$.
    Restrict $\mathbf{r}^h$: $\mathbf{r}^H = R(\mathbf{r}^h)$, where H is the next coarser level.
    Compute the coarse grid matrix $\mathbf{A}^H$
    Choose an initial guess $\hat{\mathbf{p}}^H$ for the coarse grid problem
    Step to the coarse grid: $\boldsymbol{\epsilon}^H \leftarrow$ V-Cycle$(\mathbf{A}^H, \mathbf{r}^H, \hat{\mathbf{p}}^H)$
    Interpolate and correct: $\bar{\mathbf{p}}^h \leftarrow \tilde{\mathbf{p}}^h + P(\boldsymbol{\epsilon}^H)$.
    Post-smooth with appropriate smoother:
    $\hat{\mathbf{p}}^h_{new} = postsmoothing(\mathbf{A}^h, \mathbf{f}^h, \bar{\mathbf{p}}^h)$.
end if
The solution is obtained by iterating $\mathbf{p} =$ V-Cycle$(\mathbf{A}, -\mathbf{b}, \mathbf{p})$.

Since the problem we are solving is nonlinear, the multigrid V-cycle for linear problems cannot be used to solve (4.15) directly. Specifically, the coarse grid error $\mathbf{e}^H$ cannot be evaluated through (5.17). In other words, we need to make changes regarding how to step to the coarse grid. To address this issue, we need to change the smoother and apply the Full Approximation Scheme (FAS) multigrid algorithm [10]. Firstly, we introduce the Projected Gauss-Seidel (PGS), which is based on the Gauss-Seidel method and used for

solving LCPs [25]. The idea of PGS is to enforce the non-negative constraint after each Gauss-Seidel iteration. The PGS is in general slow in solving LCP but is efficient enough to make the error smooth. The details of PGS for smoothing our LCP problem are given in Algorithm 5. In each iteration, the PGS obtain the approximate solution by performing a GS iteration. Then it checks every entry against the constraint. In case of constraint violation, the PGS enforces the constraint on the corresponding entry. For example, if the i-th entry obtained from GS iteration is negative and corresponds to the cell near the solid boundary ($i \in \mathscr{S}$), the PGS sets it to zero to enforce the constraint.

Algorithm 5 Projected Gauss-Seidel (PGS) for smoothing the LCP problem (4.15)

$\mathbf{p} \leftarrow \text{PGS}(\mathbf{A}, \mathbf{b}, \mathbf{p}, N_{max})$:
The initial guess is given as $\mathbf{p}^0 = \mathbf{p}$.
for $k = 0, 1, 2, ..., N_{max}$ **do**
  for $i = 0, 1, 2, ..., n-1$ **do**
    $r_i^k = -\mathbf{b}_i - \sum_{j \neq i} \mathbf{A}_{i,j} \mathbf{p}_j^k$
    if $i \in \mathscr{S}$ **then**
      Enforce the constraint:
      $\mathbf{p}^{k+1} = \max(0, \frac{r_i^k}{\mathbf{A}_{i,i}})$
    else
      $\mathbf{p}^{k+1} = \frac{r_i^k}{\mathbf{A}_{i,i}}$
    end if
  end for
end for
Return $p^{N_{max}}$

We use $\mathcal{N}$ to define the operator on $\mathbf{p}$ on the left hand side of (4.16) and the equation becomes $\mathcal{N}(\mathbf{p}) = 0$. Let $\mathcal{N}^h$ denote the nonlinear operator $\mathcal{N}$ on the grid at level h, where h is the grid size. According to (4.16), we have

$$\mathcal{N}^h(\mathbf{p}^h) = \inf_{\Lambda^h \in \mathscr{S}^h} \{\Lambda(\mathbf{A}^h \mathbf{p}^h + \mathbf{b}^h) + (I - \Lambda^h)\mathbf{p}^h\}, \tag{5.18}$$

where the matrix $\mathbf{A}^h$ is constructed at grid level h, vector $\mathbf{p}^h$ is the exact solution at the level h, vector $\mathbf{b}^h$ is restricted from the finer level, and the control set $\mathscr{S}^h$ is defined to be the same as in $\mathscr{S}$ except that the dimension of the matrices is the number of cells on the level h. To explain how FAS multigrid works, we consider the fine grid level h and its next coarser grid level H. The nonlinear problem at level h is

$$\mathcal{N}^h(\mathbf{p}^h) = \mathbf{f}^h. \tag{5.19}$$

Note that $\mathbf{f}^h = \mathbf{0}$ on the finest grid; see (4.16). The error after pre-smoothing is

$$\mathbf{e}^h = \mathbf{p}^h - \mathbf{q}^h, \tag{5.20}$$

where $\mathbf{q}^h$ is the approximate solution. The residual becomes

$$\mathbf{r}^h = \mathbf{f}^h - \mathcal{N}^h(\mathbf{q}^h) = \mathcal{N}^h(\mathbf{p}^h) - \mathcal{N}^h(\mathbf{q}^h). \tag{5.21}$$

Let R be the restriction operator from fine to coarse grid. Due to the nonlinear operator $\mathcal{N}^h$, we cannot use $\mathbf{r}^h = \mathcal{N}(\mathbf{e}^h)$ and restrict it into coarse grid as in the linear case. Instead, we rewrite (5.21) as

$$\mathcal{N}^h(\mathbf{p}^h) = \mathcal{N}^h(\mathbf{q}^h) + \mathbf{r}^h \tag{5.22}$$

and restrict it into the coarse level H:

$$\mathcal{N}^H(\mathbf{q}^H) = \mathcal{N}^H(R(\mathbf{q}^h)) + R(\mathbf{r}^h) \equiv \mathbf{f}^H, \tag{5.23}$$

where $\mathbf{q}^H$ be the exact solution to equation (5.23). The coarse grid error is computed as $\mathbf{e}^H = \mathbf{q}^H - R(\mathbf{q}^h)$. We interpolate the coarse grid error $\mathbf{e}^H$ on the fine grid and use $P(\mathbf{e}^H)$ to correct $\mathbf{q}^h$.

Algorithm 6 V-Cycle of the FAS multigrid for solving the LCP problem (4.15)

$\mathbf{q}^h \leftarrow$ V-Cycle$(\mathcal{N}^h, \mathbf{f}^h, \mathbf{q}^h)$:
Define the restriction operator R and interpolation operator P.
Determine the number of pre-smoothing n_1 and the number of post-smoothing n_2.
if h is the coarsest level **then**
  Solve $\mathcal{N}^h(\mathbf{p}^h) = \mathbf{f}^h$ with PGS.
else
  Pre-smooth with PGS:
  $\mathbf{q}^h = \text{PGS}(\mathcal{N}^h, \mathbf{f}^h, \mathbf{q}^h, n_1)$
  Compute the residual $\mathbf{r}^h = \mathbf{f}^h - \mathcal{N}^h(\mathbf{q}^h)$.
  Restrict $\mathbf{q}^h$: $\mathbf{q}^H = R(\mathbf{q}^h)$, where H is the next coarser level.
  Compute $\mathbf{f}^H = \mathcal{N}^H(\mathbf{q}^H) + R(\mathbf{r}^h)$.
  $\mathbf{q}^H \leftarrow$ V-Cycle$(\mathcal{N}^H, \mathbf{f}^H, \mathbf{q}^H)$
  Interpolate and correct: $\mathbf{q}^h \leftarrow \mathbf{q}^h + P(\mathbf{q}^H - R(\mathbf{q}^h))$.
  Post-smooth with PGS:
  $\mathbf{q}^h = \text{PGS}(\mathcal{N}^h, \mathbf{f}^h, \mathbf{q}^h, n_2)$
end if
The solution is obtained by iterating $\mathbf{p} =$ V-Cycle$(\mathcal{N}, \mathbf{0}, \mathbf{p})$.

Algorithm 6 provides the details of FAS multigrid. We use projected Gauss-Seidel (PGS) [16] as the smoother. For the coarse grid matrix construction, we choose Galerkin method [12] over the direct discretization because it gives a more accurate matrix. We modify the standard interpolation and restriction to accommodate the staggered grid, reduce the cost of constructing the coarse grid matrix, and maintain the same order of accuracy. In case of jumps near the solid boundary, we adjust the interpolation and restriction weights to improve accuracy. These modifications on the standard FAS enables convergence of our solver for the LCP and achieves better performance than the multigrid by Chentanez et al. [14].

Now we describe our modifications in detail.

5.3.1 Coarse grid matrix construction

The coarse grid matrix $\mathbf{A}^H$ is often constructed by directly discretizing the problem on the coarse grid. Since this way is cheap and simple, Chentanez et al. [14] computed the coarse grid matrix by averaging the the weights of non-solid matter and the liquid level set function from fine to coarse grids. We found that this approach is insufficiently accurate as it led to the coarse grid error not always matching the fine grid error, in turn causing slow convergence. In general, the complexity of the fluid domain can make it difficult to accurately approximate the shape through discretization on a coarse grid. For example, a bubble represented on the fine grid can disappear on the coarse grid. Therefore, to match the fine grid matrix more accurately, we propose to construct the coarse grid matrix using the Galerkin method [12]:

$$\mathbf{A}^H = R \cdot \mathbf{A}^h \cdot P, \tag{5.24}$$

where $\mathbf{A}^h$ is the fine grid matrix. At the expense of getting a much denser matrix, which results in extra computational costs, this approach ensures that the coarse grid operator accurately matches its fine grid counterpart, and gives faster convergence. In particular, this ensures that relatively thin boundaries which are only fully resolved at the finest level are still naturally respected at coarser grid levels, without additional treatment. To reduce the extra costs from using Galerkin method, we use the compact interpolation and restriction (explained later), whose operators are much sparser, to make the coarse grid matrices much sparser and faster to construct.

5.3.2 Interpolation and restriction

We describe how to construct the restriction matrix R and the interpolation matrix P. To preserve symmetry, the restriction matrix is computed as the transpose of the interpolation matrix scaled by a constant ($\frac{1}{4}$ in 2D and $\frac{1}{8}$ for 3D). A natural first choice for interpolation is bilinear in 2D and trilinear in 3D, since they are both second order.

Because we have adopted a staggered grid with pressures at cell centers, we must do it differently when interpolating between levels. The layout of two grid levels and their pressure samples in 2D is shown in Figure 5.4. The solid line represents the coarse grid and the dotted line the fine grid. Fine and coarse grid pressure samples are represented by black and red points, respectively. Bilinear interpolation for the fine grid point p^h depends on the four nearest coarse grid points p_0^H, p_1^H, p_2^H, p_3^H. Their interpolation weights are $\frac{9}{16}$, $\frac{3}{16}$, $\frac{3}{16}$, $\frac{1}{16}$, respectively. The trilinear interpolation in 3D is shown in Figure 5.5: p^h is interpolated from eight surrounding coarse grid points with weights of $\frac{27}{64}$ for p_0^H, $\frac{9}{64}$ for p_1^H, p_2^H, p_4^H, $\frac{3}{64}$ for p_3^H, p_5^H, p_6^H, and $\frac{1}{64}$ for p_7^H.

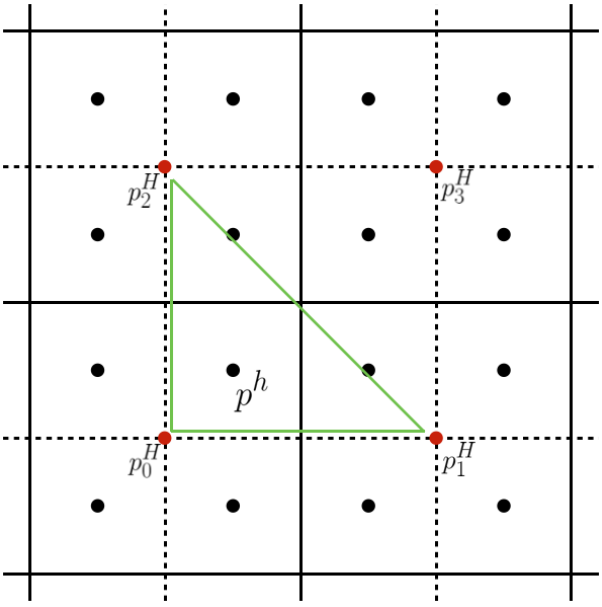


Figure 5.4: *Interpolation of pressure between grid levels in 2D.*

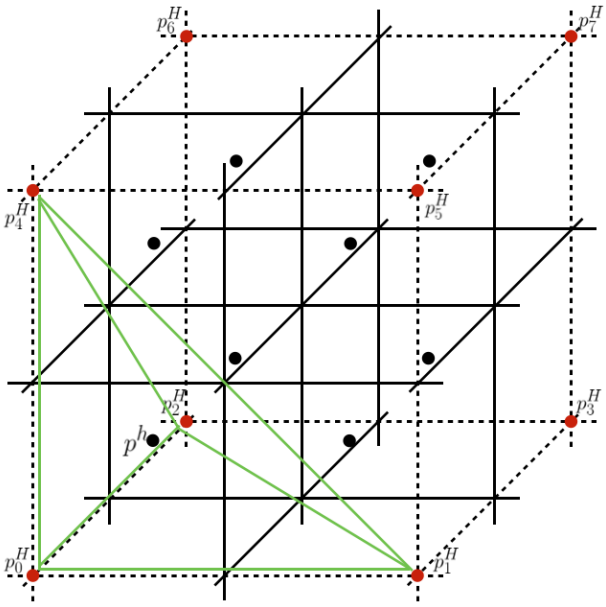


Figure 5.5: *Interpolation of pressure between grid levels in 3D.*

As we mentioned earlier, the coarse grid matrix is constructed by multiplying the interpolation and restriction matrices. As a result, the density of the coarse grid matrix depends on how we perform the interpolation and restriction. Using too many nearby coarse grid points yields denser interpolation and restriction matrices therefore a denser coarse grid matrix, which makes the computation more expensive. Thus, we propose an even simpler alternate solution that uses fewer coarse grid points while preserving the desired second order accuracy. Specifically, we adopt the barycentric interpolation, which gives a sparser interpolation operator. As a result, the coarse grid matrix is sparser and the construction is less expensive. Moreover, the smoothing process on coarse grids will be less expensive due to the decrease of nonzeros per row. In our experiments in 3D, both the average number of nonzeros per row of the top level coarse grid matrix and the time for solving the LCP were reduced by about half.

We now describe how to perform the barycentric interpolation. Consider again p^h in Figure 5.4 for the 2D case, we will use the coarse grid points p_0^H, p_1^H, p_2^H, since they form a triangle that includes the non-coarse grid point p^h (green lines in Figure 5.4). The interpolation becomes $p^h = \frac{1}{2}p_0^H + \frac{1}{4}p_1^H + \frac{1}{4}p_2^H$. Similarly, in 3D as shown in Figure 5.5, p^h is inside the tetrahedron (green lines in Figure 5.5) formed by p_0^H, p_1^H, p_2^H, p_4^H. Each coarse grid point is assigned a weight of $\frac{1}{4}$. The weights for 2D and 3D are determined such that second order accuracy is preserved. We will give detailed proof later.

These more compact interpolation (and corresponding restriction) operators give interpolation matrices that will be 25 percent sparser in 2D and 50 percent sparser in 3D. We give a 3D example with small problem size (from 8^3 to 4^3) shown in Figure 5.6 to visualize how much sparser can be achieved when constructing coarse grid matrix through barycentric interpolation and restriction. The coarse grid matrix from trilinear and barycentric has 2,104 and 1,168 non-zeros, respectively. The operator P from trilinear and barycentric has 2,744 and 1,664 non-zeros, respectively. For larger problem size (from 64^3 to 32^3) shown in Figure 5.7, the number of nonzeros demonstrates that using the barycentric makes the coarse grid matrix over 50 percent sparser (858,304 vs 2,000,376 non-zeros). The operator P from barycentric (1,024,000 non-zeros) is also 50 percent sparser than from trilinear (2,000,376 non-zeros).

We prove that the barycentric interpolation gives accuracy of second order. For simplicity, let's assume the cube in Figure 5.5 is $[0,h]\times[0,h]\times[0,h]$ and we want to interpolate the pressure at $(\frac{h}{4},\frac{h}{4},\frac{h}{4})$ with the barycentric interpolation:

$$p^h = \frac{1}{4}p_0^H + \frac{1}{4}p_1^H + \frac{1}{4}p_2^H + \frac{1}{4}p_4^H, \tag{5.25}$$

where $p_0^H = p(0,0,0)$, $p_1^H = p(h,0,0)$, $p_2^H = p(0,h,0)$, $p_4^H = p(0,0,h)$. Assume the pressure p is C^1 smooth and apply Taylor expansion to p_0^H, p_1^H, p_2^H, p_4^H at $(\frac{h}{4},\frac{h}{4},\frac{h}{4})$:

$$p_0^H = p\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial x}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\left(-\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial y}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\left(-\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial z}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\left(-\tfrac{h}{4}\right) + O(h^2), \tag{5.26}$$

$$p_1^H = p\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial x}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\tfrac{3h}{4} + \tfrac{\partial p}{\partial y}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\left(-\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial z}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\left(-\tfrac{h}{4}\right) + O(h^2), \tag{5.27}$$

$$p_2^H = p\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial x}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\left(-\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial y}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\tfrac{3h}{4} + \tfrac{\partial p}{\partial z}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\left(-\tfrac{h}{4}\right) + O(h^2), \tag{5.28}$$

$$p_4^H = p\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial x}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\left(-\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial y}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\left(-\tfrac{h}{4}\right) + \tfrac{\partial p}{\partial z}\left(\tfrac{h}{4},\tfrac{h}{4},\tfrac{h}{4}\right)\tfrac{3h}{4} + O(h^2). \tag{5.29}$$

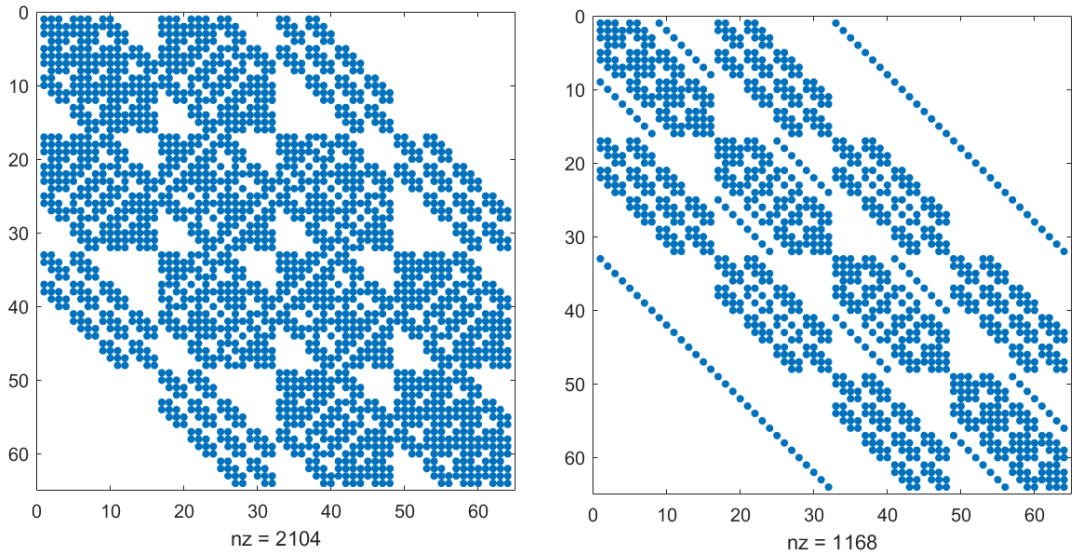


Figure 5.6: *Visualization of the coarse grid matrix for 3D problem computed using trilinear (left) and barycentric (right) interpolation/restriction from* 8^3 *to* 4^3.

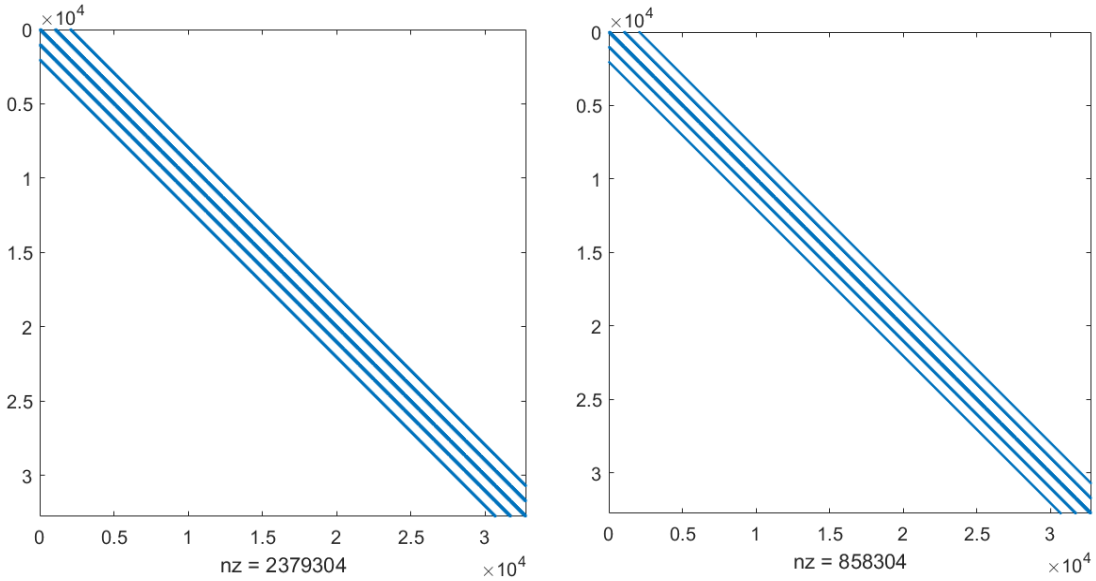


Figure 5.7: *Visualization of the coarse grid matrix for 3D problem computed using trilinear (left) and barycentric (right) interpolation/restriction from* 64^3 *to* 32^3.

Adding equations (5.26) to (5.29) together and dividing both sides by 4, we find that all the first derivative terms are cancelled. Therefore, we have the error:

$$p^h - p(\frac{h}{4}, \frac{h}{4}, \frac{h}{4}) = O(h^2), \tag{5.30}$$

which concludes the proof in 3D. The proof in 2D is similar.

Finally, we present the experimental results in 3D (see Table 5.1) to show the performance gain from using barycentric interpolation and restriction in Galerkin method. Compared with direct discretization, the Galerkin method with trilinear is much more computationally expensive (at least 7 times slower). However, using barycentric greatly reduces the coarse grid matrix construction time and is even able to achieve similar performance as direct discretization for large problems (eg. 128^3).

Coarse grid matrix size	Direct discretization	Galerkin method (trilinear)	Galerkin method (barycentric)
16^3	0.00	0.03	0.01
32^3	0.01	0.22	0.04
64^3	0.14	1.71	0.31
128^3	2.55	14.01	2.56

Table 5.1: *Coarse grid matrix construction timing (in seconds) comparison in 3D*

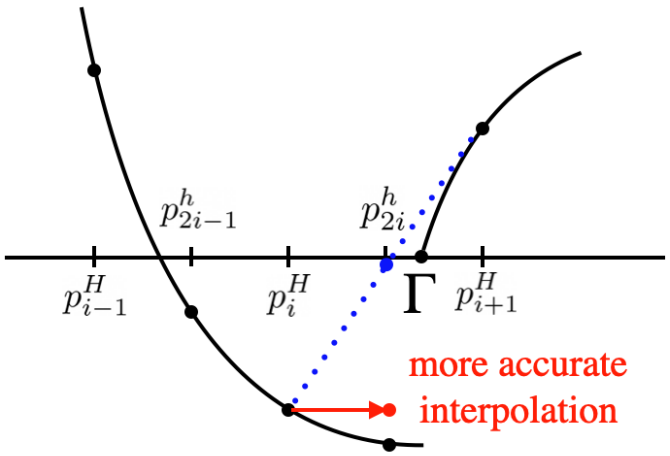


Figure 5.8: *Standard interpolation (dashed blue) across a narrow solid boundary causes large pressure errors (shown in 1D). Our one-sided interpolation (red) yields better behavior.*

5.3.3 Boundary handling

The solution to the pressure equation (4.15) may have a large jump around the solid boundary due to different conditions being imposed on either side of it (see Figure 5.8). Therefore, we modify our interpolants to accommodate this situation and ensure better convergence of FAS multigrid. Inspired by the work of Wan & Liu [60] and Guendelman et al. [36], we modify our interpolants to exploit knowledge of the solid boundary. Given a fine grid point, we determine the interpolation weights for only the coarse grid points which are on the same side of the solid. We then rescale the weights of those (same side) coarse grid points proportionally so that they sum to 1. For example, consider a 2D coarse grid cell crossing the boundary as shown in Figure 5.9.

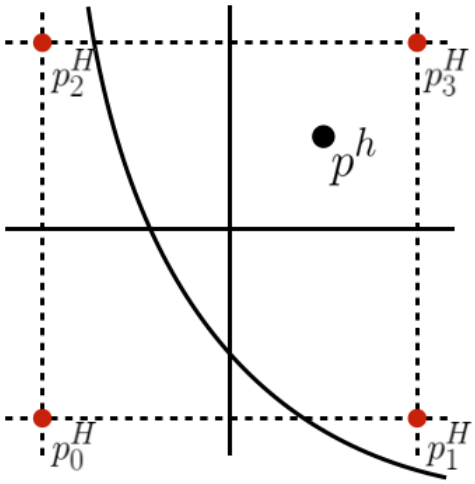


Figure 5.9: *Interpolation near the solid boundary in 2D.*

Points p_0^H and p_2^H are on the opposite side of the solid from the find grid point p^h. Barycentric interpolation ordinarily assigns weights of $\frac{1}{4}$, $\frac{1}{4}$, $\frac{1}{2}$ to p_1^H, p_2^H, p_3^H, respectively. In our modified one-sided interpolation, only points p_1^H and p_3^H will be used; their weights after rescaling become $\frac{1}{3}$ and $\frac{2}{3}$, so we have $p^H = \frac{1}{3}p_1^H + \frac{2}{3}p_3^H$.

For the linear multigrid used for the inner linear systems in our policy iteration and penalty method, we likewise use barycentric interpolation/restriction and neglect points across the solid boundary, but find it unnecessary to normalize the weights.

Chapter 6

Numerical results

In this chapter, we demonstrate efficiency and scalability of our LCP solvers by presenting numerical results from simulating several scenarios in both 2D and 3D using separating solid boundary conditions. We compare the performance of our solvers with the recent competitive methods: the FMG [14] and the Newton's method [3]. The comparison is based on scalability, timing, and convergence.

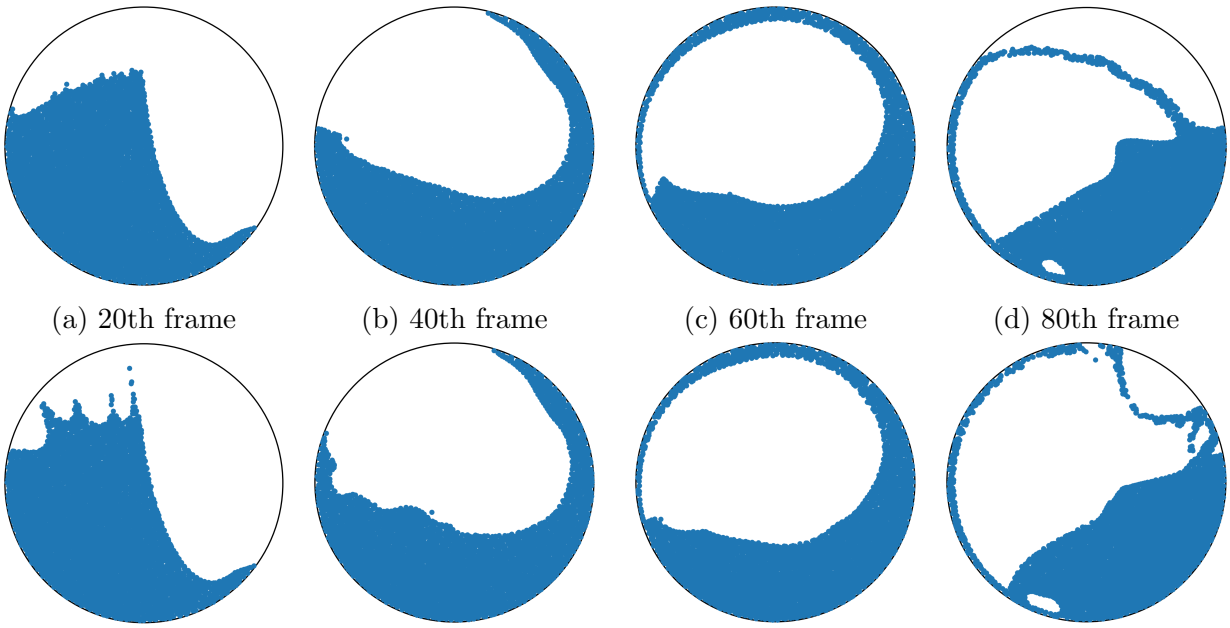
6.1 Environmental set-up and parameters

The fluid simulation tool we are using in 2D and 3D is implemented by Batty [6, 7] written in C++ . The domain is $[0,1]^2$ for 2D and $[0,1]^3$ for 3D. The time step for each frame is 0.01. We use an absolute residual tolerance of 10^{-6} throughout. To ensure convergence of the penalty method, the penalty term ρ must be sufficiently large, which depends on the desired tolerance. In our experiments, we chose ρ to be 10^3/tolerance (i.e., 10^9). For all the multigrid methods, we choose the coarsest grid as 4^2 in 2D and 4^3 in 3D for all problem sizes.

All the experiments except the maze scenario in Figure 6.7 were run on the slave nodes of the Chardonnay cluster consisting of 2 master nodes and 8 slave nodes. Each slave node has 2x Intel E5-2670 (8C) CPUs, 128 GB memory, 15T LSI SAS2308 disk space, 2x GbE interconnect, and X9DRFF-7TG+ motherboard. The operating system is Linux Ubuntu 20.04.1 LTS and the C++ code is compiled with optimization option O2.

As for the maze scenario, the code for the simulation is provided by the developers of Newton's method [24], which needs to be compiled with CUDA and offers the option of

running either on CPU or GPU. Since the Chardonnay cluster does not have GPU, we use the node from the Graham cluster [1] to get the code compiled and run the experiments. The node we use has 2 x Intel Xeon Silver 4110 Skylake @ 2.10GHz CPUs, 192 GB memory, 11.0TB SATA SSD storage, and 4 x NVIDIA T4 Turing GPUs. We run the code on CPU for the comparison.



(a) 20th frame (b) 40th frame (c) 60th frame (d) 80th frame

(e) 20th frame (f) 40th frame (g) 60th frame (h) 80th frame

Figure 6.1: *Snapshots from simulating fluid inside a solid circle in 2D using LCP boundaries (top row) and standard boundaries (bottom row).*

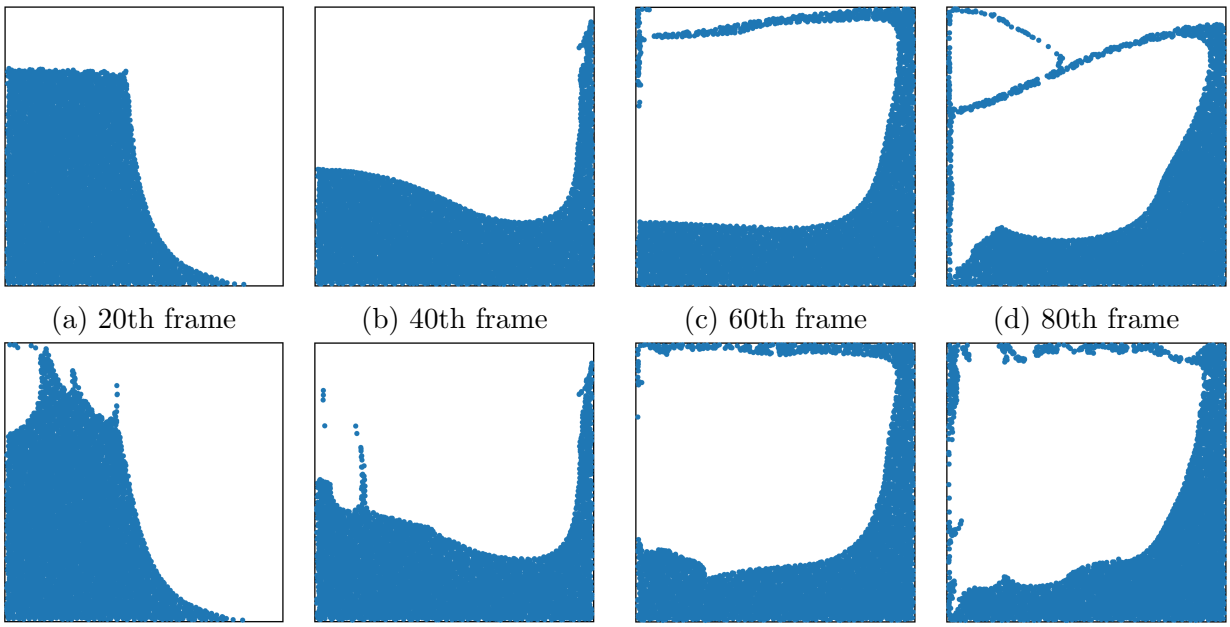
6.2 Simulation scenarios

We present several scenarios in both 2D and 3D demonstrating the effects of using separating solid boundary conditions along with representative frames. We will run simulations on these scenarios to measure the performance of LCP solvers.

Figures 6.1 and 6.2 show frames from two scenarios in 2D which demonstrate the difference between standard and LCP solid boundary conditions. The scenario in Figure 6.1 simulates half fluid on the left inside a circular solid boundary in 2D. As we can see, at the 20th frame, the fluid naturally slips from the top boundary when using separating

solid boundary conditions. However, for the standard boundary conditions, the fluid does not fall smoothly as if it is sticky. We can also see from the 40th frame that the fluid using LCP boundaries is more tentative to peel off from the solid near the left most point. From the 60th frame to the 80th frame, we can see that the fluid with separating solid boundary conditions quickly drops from the ceiling while the fluid with standard boundary conditions sticks to the top boundary. The scenario in Figure 6.2 simulates half fluid on the left inside a solid square in 2D, which shows similar effects as the solid circle scenario. The fluid can easily slip from the top (20th, 60th, and 80th frames) and the left (40th frame) boundaries with separating solid boundary conditions, compared with the standard boundary conditions.

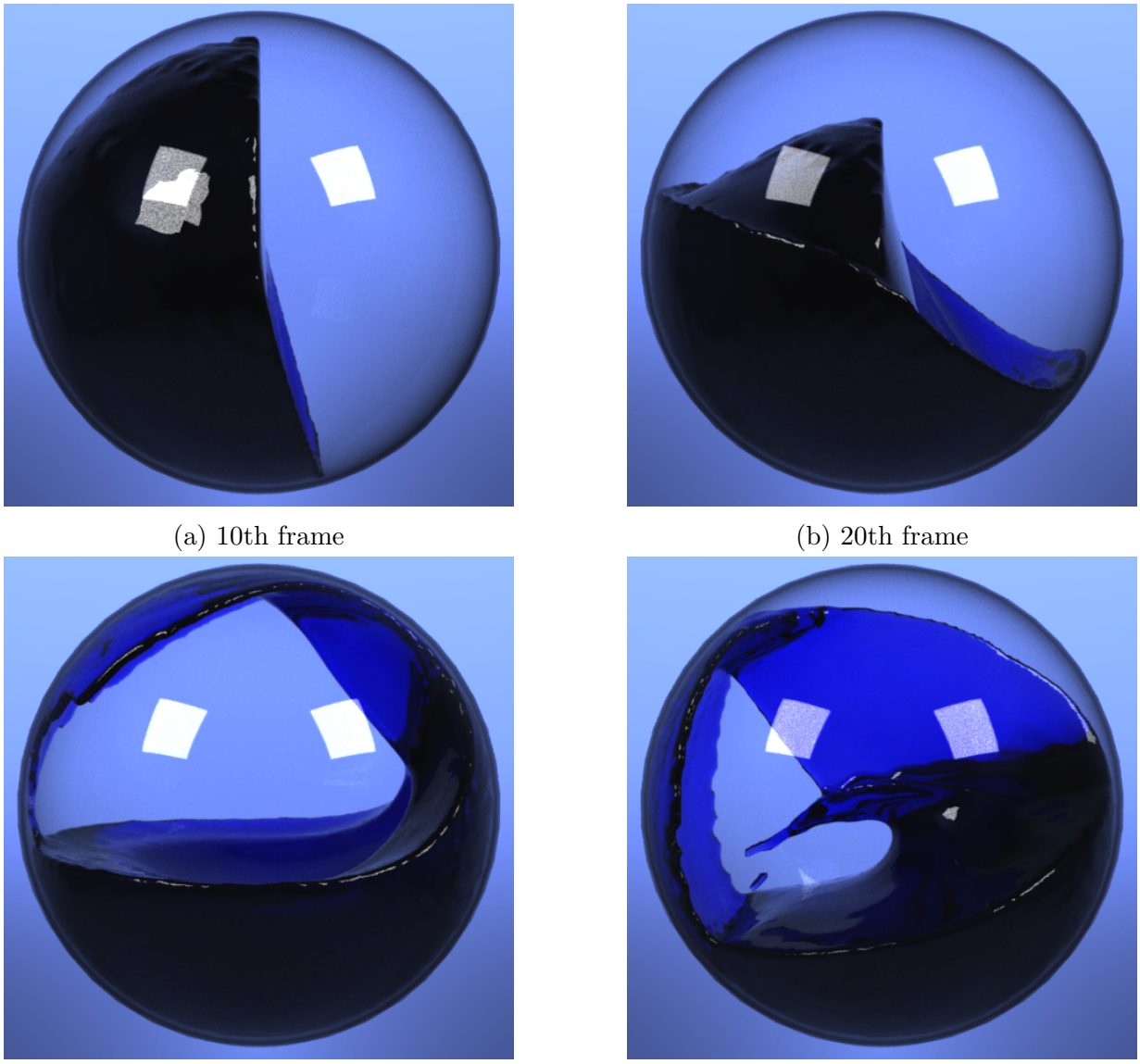
We look at three scenarios in 3D. At first, we present selected frames from the scenario 1 in 3D in Figures 6.3 for separating solid boundary conditions and 6.4 for standard boundary conditions. The scenario 1 simulates half fluid filling the left space inside a spherical boundary. The 20th frame shows that the fluid drops from the boundary for the LCP case instead of sliding around it. It is also shown in the other frames that the fluid more readily separates from the boundary in the LCP case.



(a) 20th frame (b) 40th frame (c) 60th frame (d) 80th frame

(e) 20th frame (f) 40th frame (g) 60th frame (h) 80th frame

Figure 6.2: *Snapshots from simulating fluid inside a solid square in 2D using LCP boundaries (top row) and standard boundaries (bottom row).*



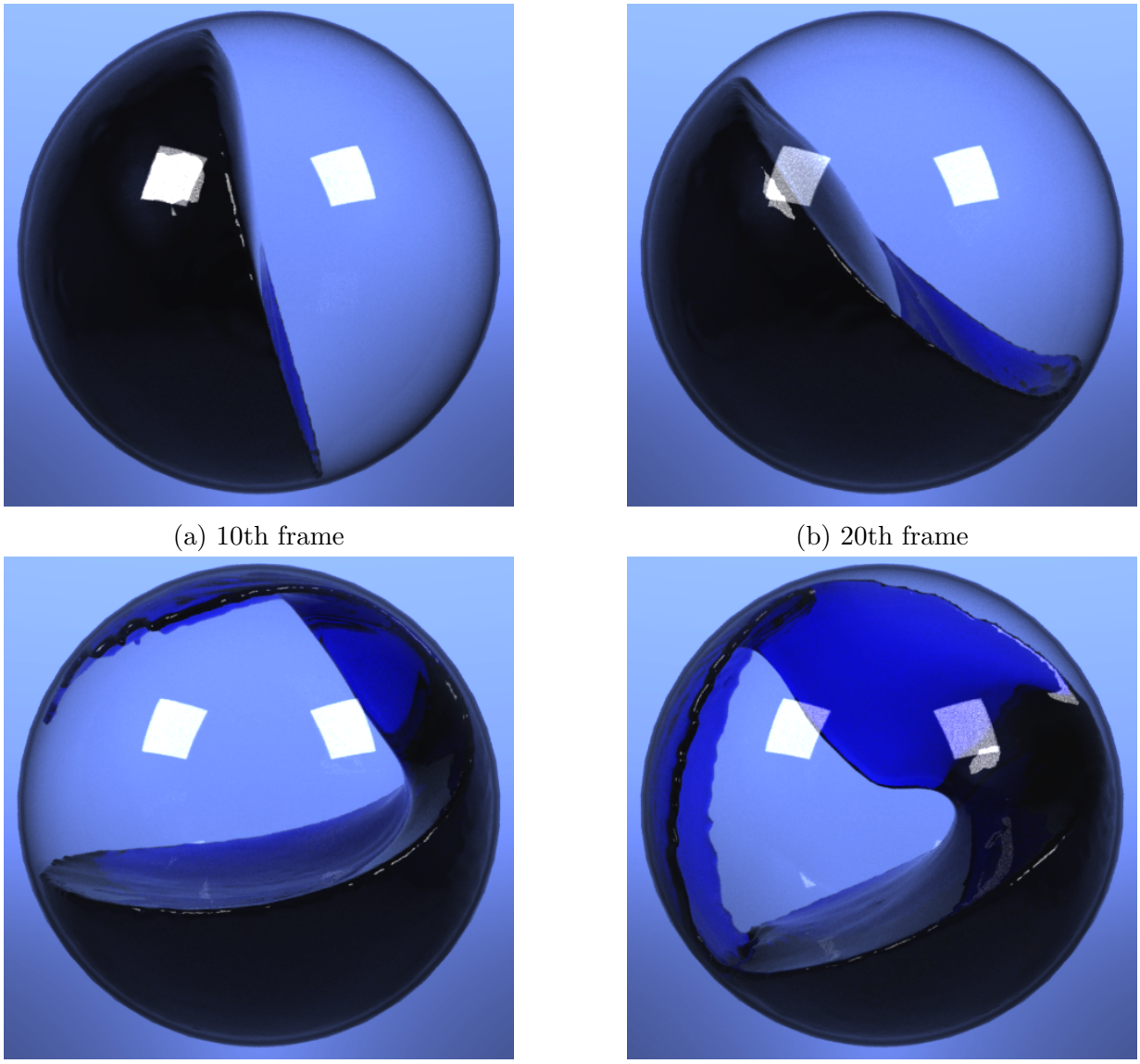
(a) 10th frame

(b) 20th frame

(c) 50th frame

(d) 70th frame

Figure 6.3: *Snapshots from simulating fluid inside a solid sphere in 3D at* 128^3 *using LCP boundaries.*



(a) 10th frame

(b) 20th frame

(c) 50th frame

(d) 70th frame

Figure 6.4: *Snapshots from simulating fluid inside a solid sphere in 3D at* 128^3 *using standard boundaries.*

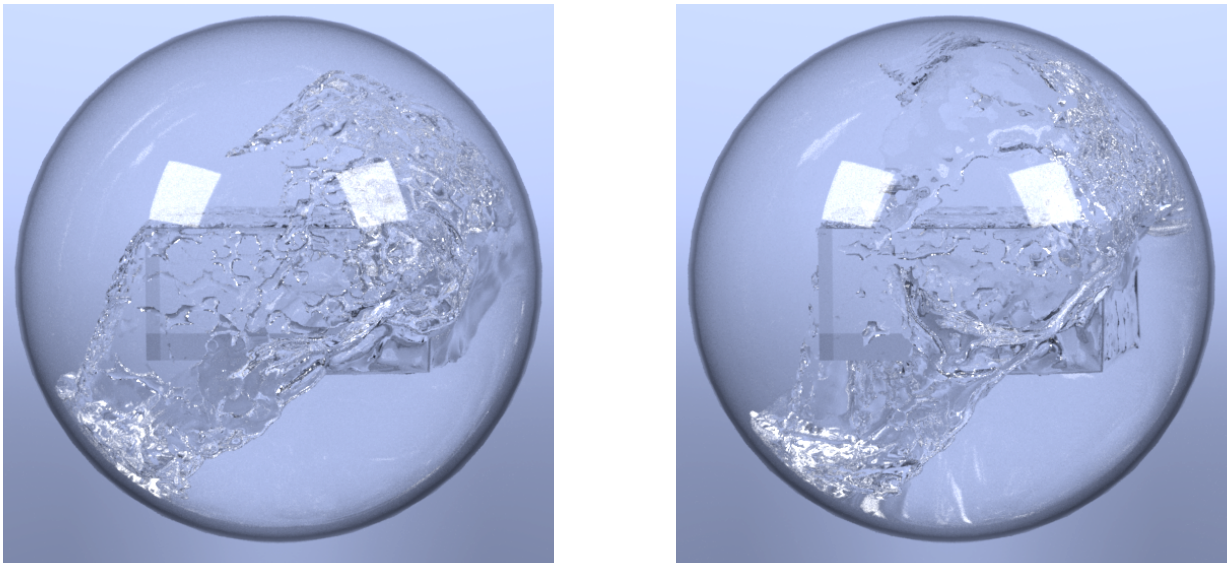


Figure 6.5: *The 70th frame from scenario 2 in 3D, with (left) and without (right) separating solid wall boundary conditions.*

We now present more complicated scenarios. Scenario 2 is based on scenario 1 but has a rectangular cuboid inside so that fluid-solid contacts frequently form and break on the cuboid's bottom face. At the 70th frame shown in Figure 6.5, we can see that the water drops from the top with separating solid boundary conditions while it sticks to the the boundary when using standard boundary conditions. Scenario 3 has a solid outer boundary created as the union of two spheres. Two hollow solid spheres are also placed inside the domain with the bottom one having holes through it. The effects of using separating solid boundary conditions for scenario 3 is demonstrated in Figure 6.6. At the 10th frame, the water drops from top for LCP case instead of sliding along the boundary.

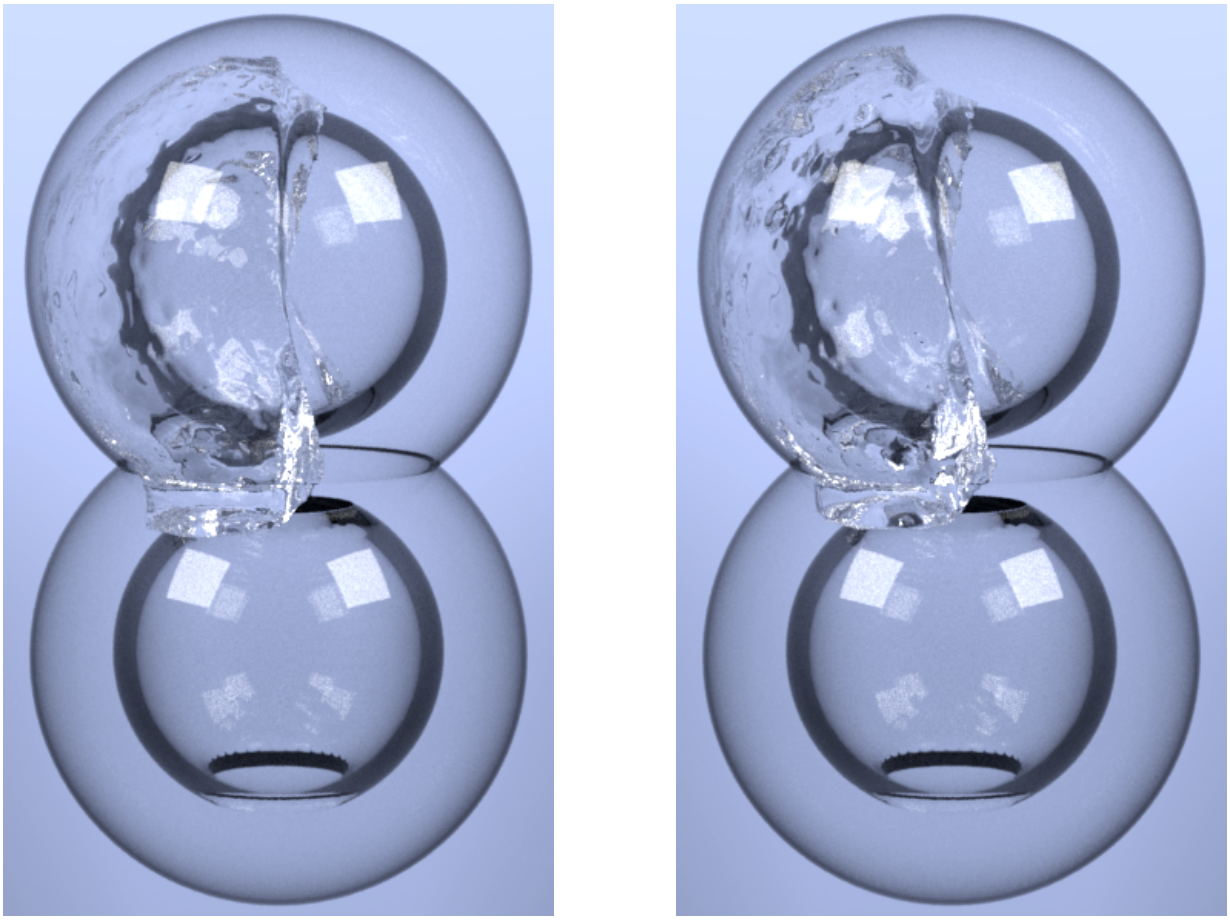


Figure 6.6: *The 10th frame from scenario 3 in 3D, with (left) and without (right) separating solid wall boundary conditions.*

6.3 Performance analysis

To evaluate the effectiveness of our proposed solvers, we compare the performance of different solvers, including the standard boundary conditions using PCG (No LCP). Specifically, we compare our methods with the multigrid method developed by Chentanez et al. [14] and the non-smooth Newton's method [3]. Chentanez's full multigrid (FMG) approach iterates costly full cycles while our multigrid uses V-cycles, which are simpler and less expensive. Moreover, they pre- and post-smooth the error four times, while we do it only twice. For

policy iteration, we tested with two different solvers for the linear system (5.8): PCG and (linear) multigrid, denoted by PI-PCG and PI-MG, respectively. The coarse grid matrix here was also computed using the Galerkin method. We likewise tested penalty method using PCG (PE-PCG) and multigrid (PE-MG). Newton's method is broadly similar in concept to policy iteration, but is much more complicated. In addition to solving a linear system in each Newton iteration, it also requires performing a line search. For the purpose of testing and comparing with Newton's method, we also use their public CUDA code [24] and ran tests using the CPU (rather than GPU). We measure the performance of these methods based on the scalability and the timing. For multigrid methods, we show their convergence at some representative frames in the simulations. Finally, we demonstrate the contribution of our proposed modifications to the FAS-MG by comparing with the standard FAS-MG.

6.3.1 Scalability

To compare scalability, we run the simulations for the scenarios in both 2D and 3D. Specifically, we choose the scenario in 2D as shown in Figure 6.1 and in 3D as shown in Figures 6.3 and 6.4.

We measure the scalability using the average number of iterations per pressure equation. Specifically, this refers to the average number of V-cycles per pressure equation over 100 frames for FAS-MG, and average number of full cycles for FMG. The solution processes of policy iteration, penalty method and Newton's method involve nested iterations, often called outer-inner iteration. The outer iteration updates the linear system whereas the inner iteration solves the linear system by an iterative method. We count all the inner iterations per pressure equation. We measure iterations for policy iteration and penalty method as the total number of PCG/MG iterations per pressure equation. For Newton's method, we add the number of PCG and line search iterations together to count as the number of iterations.

Size	FAS-MG	PI-PCG	PI-MG	PE-PCG	PE-MG	FMG	Newton	No LCP
32	7.45	16.9	7.59	16.87	7.99	20.38	92.11	15.99
64	10.39	36.02	9.74	36.07	12.58	35.4	336.17	30.73
128	14.07	70.63	11.89	70.58	17.6	65.37	947.23	55.91
256	18.26	123.61	14.51	122.42	21.98	109.54	2283.65	93.2

Table 6.1: *Average number of iterations per pressure equation for solving 100 frames of the circular domain problem in 2D.*

Size	FAS-MG	PI-PCG	PI-MG	PE-PCG	PE-MG	FMG	Newton	No LCP
32	9.75	29.9	12.12	30.16	13.93	22.21	1901.69	20.14
64	10.44	70.92	14.66	71.34	19.11	39.81	3890.08	39.57
128	11.94	132.56	17.61	133.53	24.57	61.12	10879.01	62.26
256	14.02	251.88	20.71	252.28	28.84	89.89	NA	100.78

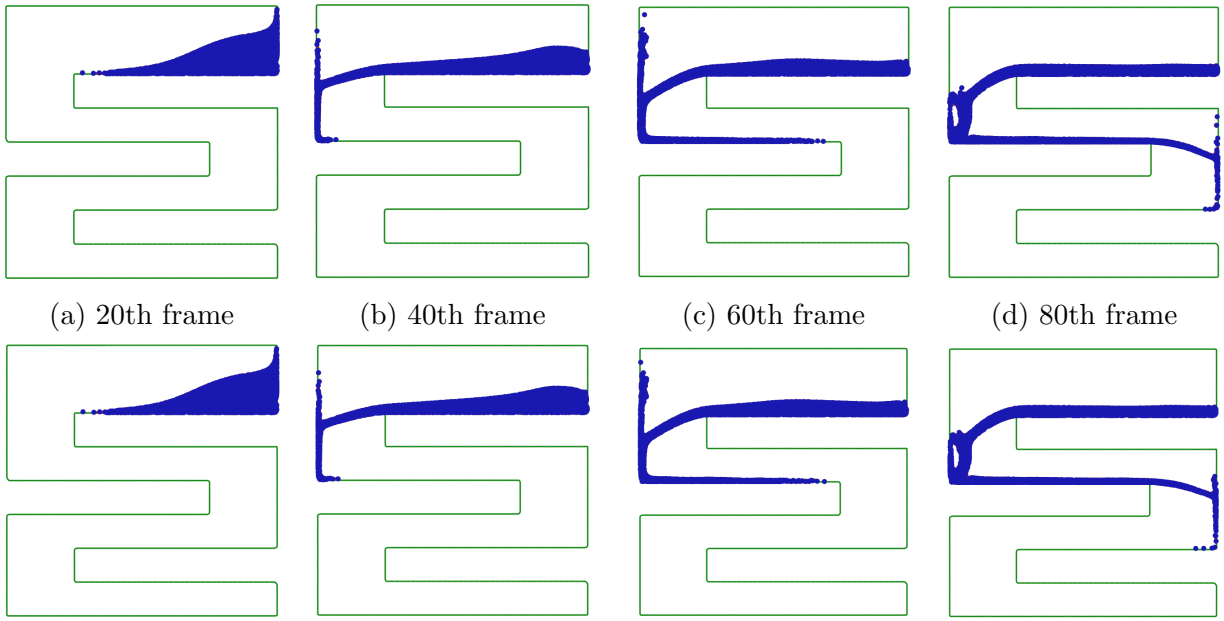
Table 6.2: *Average number of iterations per pressure equation for solving 100 frames of the spherical domain problem in 3D.*

We perform the tests for different problem sizes, namely, the numbers of unknowns per dimension: 32, 64, 128, and 256. For the 256^3 grid in 3D, we did not test Newton's method because it was too slow. The test results for 2D and 3D are shown in Table 6.1 and Table 6.2, respectively. We recall that a method is considered to be scalable if the number of iterations does not depend on the problem size.

For the 2D test, we can see that for our multigrid methods (FAS-MG, PI-MG, PE-MG), the average number of iterations only increases 2 to 3 times from size of 32 to 256. For the FMG, however, the number increases about 5 times. For the policy iteration and penalty method with PCG, the average number of iterations increases 10 times from size of 32 to 256. The FMG does demonstrate better scalability than the policy iteration and penalty method with PCG but is not as good as our multigrid methods. This is because the PCG solver itself is not scalable. The Newton's method has much worse scalability as the average number of iterations increases about 24 times from size of 32 to 256. For the No LCP, the average number of iterations increases about 6 times from size of 32 to 256, which scales better than PI-PCG and PE-PCG but worse than the multigrid methods.

For the 3D test, the result is similar to 2D but our multigrid methods have slightly better scalability. The average number of iterations increase 1 to 2 times from size of 32 to 256. The other methods scale more or less the same as in the 2D case.

As an additional comparison against Newton's method [3], we used the authors' code and tested their method and all of our LCP solvers in their maze scenario shown in Figure 6.7. Due to the limitation of their code, we only present the test for the problem sizes up to 128 in 2D. The test result is shown in Table 6.3.



(a) 20th frame (b) 40th frame (c) 60th frame (d) 80th frame

(e) 20th frame (f) 40th frame (g) 60th frame (h) 80th frame

Figure 6.7: *Snapshots from simulating fluid inside a maze in 2D using our FAS-MG (top row) vs. the non-smooth Newton's method (bottom row). The results are visually consistent.*

Size	FAS-MG	PI-PCG	PI-MG	PE-PCG	PE-MG	FMG	Newton
32	4.41	9.74	8.11	9.5	7.67	6.89	87.2
64	7.07	15.52	10.55	15.75	10.11	10.71	195.5
128	5.83	31.29	13.52	31.9	12.91	34.13	450.21

Table 6.3: *Comparison of the number of iterations between our solvers and Newton's method for solving the pressure equations for 100 frames of the maze problem in 2D.*

The LCP solvers in the maze scenario (Table 6.3) scales similarly as the half fluid scenarios in 2D (Table 6.1) and 3D (6.2). However, the PI-PCG and PE-PCG have better scalability in the maze scenario. The simulation result also shows that our solver FAS-MG produces consistent result as the Newton's method (see Figure 6.7).

Size	FAS-MG	PI-PCG	PI-MG	PE-PCG	PE-MG	FMG	Newton
32	7	139	10	146	21	29	2773
64	8	282	9	301	18	61	5837
128	8	593	10	657	24	128	18680
256	9	1263	14	1255	28	234	38547

Table 6.4: *Number of iterations for solving the pressure at the 10th frame of scenario 1 in 3D.*

Size	FAS-MG	PI-PCG	PI-MG	PE-PCG	PE-MG	FMG	Newton
32	6	149	11	161	18	35	2634
64	7	308	18	336	29	43	11122
128	8	503	22	548	40	63	50162
256	9	1186	43	1313	89	103	95823

Table 6.5: *Number of iterations for solving the pressure at the 70th frame of scenario 2 in 3D.*

Size	FAS-MG	PI-PCG	PI-MG	PE-PCG	PE-MG	FMG	Newton
32	7	80	14	85	23	15	1430
64	15	182	37	203	57	43	9219
128	9	421	34	446	90	67	46491
256	12	762	24	887	60	175	122080

Table 6.6: *Number of iterations for solving the pressure at the 10th frame of scenario 3 in 3D.*

Now we test LCP solvers for specific frames with significant handling of fluid solid separation in three different scenarios in 3D (see the 10th frame for scenario 1 in Figures 6.3 and 6.4, the 70th frame for scenario 2 in Figure 6.5, the 10th frame for scenario 3 in Figure 6.6, respectively.) We choose these frames as they are representative in terms of demonstrating the effects of using separating solid boundary conditions. The number of iterations for solving the pressure for the first substep of the specific frame in the three

different scenarios are presented in Tables 6.4, 6.5, 6.6, respectively. Table 6.4 shows the scalability of LCP solvers at the 10th frame is similar to the scalability measured over all frames (Table 6.2). Compared with scenario 1, the tests in scenarios 2 and 3 demonstrate that the FAS-MG is still scalable even for complicated scenarios. The scalability of PI-MG and PE-MG deteriorates a little bit, but they still perform better than the existing methods (FMG and Newton).

Regarding scalability in terms of number of iterations, our FAS-MG, PI-MG and PE-MG are scalable because the number of iterations increases slowly with increasing problem size. Our FAS-MG scales the best since it has no outer iterations. Our PI-MG and PE-MG are scalable due to the use of multigrid method and the good scalability of the number of outer iterations of policy iteration.

We now explore the scalability of policy iteration, penalty method and Newton's method in terms of the number of outer iterations. We measure the average number of outer iterations for these methods from size of 32 to 256 over 100 frames for the half fluid scenarios in 2D and 3D are shown in Tables 6.7 and 6.8, respectively.

Size	PI-PCG	PI-MG	PE-PCG	PE-MG	Newton
32	1.12	1.12	1.12	1.23	0.94
64	1.25	1.25	1.25	1.43	1.71
128	1.34	1.34	1.35	1.58	2.08
256	1.43	1.43	1.41	1.63	2.34

Table 6.7: *Average number of* outer *iterations per pressure equation for policy iteration and penalty method for solving 100 frames of the circular domain problem in 2D.*

Size	PI-PCG	PI-MG	PE-PCG	PE-MG	Newton
32	1.47	1.47	1.49	1.53	2.04
64	1.78	1.76	1.79	1.77	2.57
128	2.1	2.12	2.12	2.19	2.91
256	2.48	2.52	2.48	2.49	NA

Table 6.8: *Average number of* outer *iterations per pressure equation for policy iteration and penalty method for solving 100 frames of the spherical domain problem in 3D.*

Tables 6.7 and 6.8 show that the number of outer iterations for policy iteration and the penalty method is almost constant with only 2 to 3 linear systems needed per pressure

equation. Policy iteration performs slightly better than the penalty method in terms of average number of outer iterations. Moreover, our policy iteration and penalty method scale slightly better than Newton's method and require fewer outer iterations in general. We note that the result for PI-PCG (or PE-PCG) is slightly different from PI-MG (PE-PCG) due to round-off errors in numerical computation. They are supposed to be the same theoretically.

The scalability of both inner and outer iterations contributes to the scalability of PI-MG and PE-MG. However, for PI-PCG and PE-PCG, the number of iterations doubles as the problem size doubles. This is expected as the number of PCG iterations usually doubles with problem size, while the number of outer iterations remains relatively constant. Newton's method is comparatively slow, especially in 3D, because its inner iteration is computationally expensive although its outer iteration is relatively scalable. FMG also suffers from an increase in the number of iterations for full cycle although the rate is less than 2 (about 1.5). We note that the number of iterations required to converge for FMG is much larger than for our proposed multigrid methods (FAS-MG, PI-MG, and PE-MG) because it is not designed for nonlinear problems. FAS-MG and PI-MG required the smallest number of iterations among all the methods.

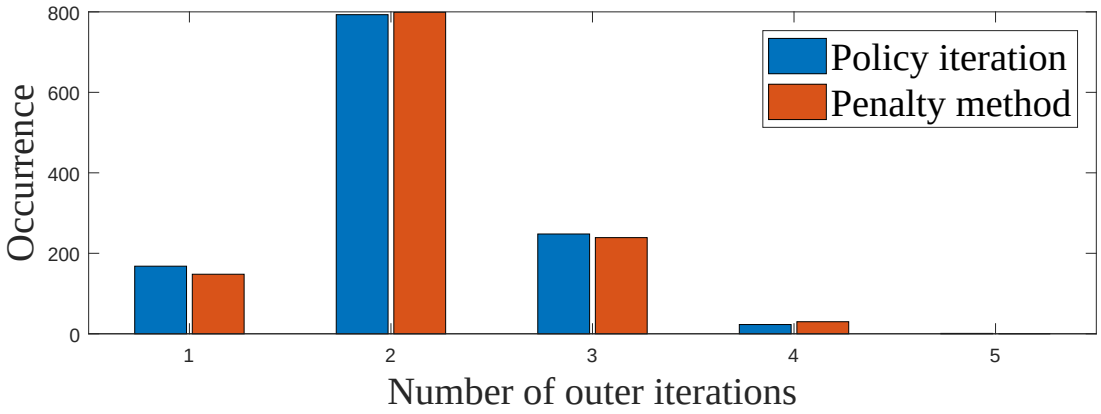


Figure 6.8: *A histogram illustrating how many pressure solves required a given number of outer iterations for policy iteration and penalty method for 100 frames of the spherical domain problem in 3D for grid size 128.*

We observed that the pressure equations for most timesteps required solving the LCP equations (i.e., performing multiple outer iterations) when the problem size becomes large. We present a histogram of the number of outer iterations for policy iteration and penalty

method over 100 frames in the 3D test for the size of 128 in Figure 6.8. About 93 percent of these pressure equations are LCP problems while some of them need only one outer iteration because the first control update leads to the correct linear system to solve the LCP. Since we use sub-stepping, the number of problems solved exceeds the frame count.

6.3.2 Timing

We present the timing comparison for the scenarios mentioned at the beginning of section 6.3.1, namely, half fluid scenarios in 2D (Figure 6.1) and in 3D (Figures 6.3 and 6.4). We measure the average time for solving the pressure equations over 100 simulation frames. Tables 6.9 and 6.10 show the average time per pressure equation for all methods in 2D and 3D, respectively. Table 6.9 shows that in 2D our proposed methods (FAS-MG, PI-PCG, PI-MG, PE-PCG, PE-MG) have similar performance especially for large grid sizes (128 and 256) and are much faster than FMG and Newton's method. No LCP is the fatest but it is expected as it solves a less complicated problem than LCP solvers. However, for the size of 256, our LCP solvers are only slightly slower than the No LCP solver. From Table 6.10, we observe that like in 2D, our proposed methods are still much faster than FMG and Newton's method in 3D. However, the performance of our proposed methods are different when the size is 256. The FAS-MG performs much better than the other proposed solvers. It is as much as 30 times faster than FMG in 3D at 256^3, and is only slightly slower than the No-LCP solver. We also present the timing for the maze scenario in 2D, shown in Table 6.11. It also demonstrates superior efficiency of our methods over FMG and Newton's method although the our proposed methods perform differently. The PI-PCG/PE-PCG perform the best but we note that the largest problem size is only 128.

Size	FAS-MG	PI-PCG	PI-MG	PE-PCG	PE-MG	FMG	Newton	No LCP
32	0.0021	0.00054	0.0028	0.00055	0.0025	0.0026	0.002	0.00048
64	0.0082	0.0034	0.0086	0.0033	0.0075	0.022	0.024	0.0029
128	0.029	0.02	0.03	0.021	0.028	0.17	0.24	0.016
256	0.13	0.15	0.12	0.13	0.13	1.25	2.59	0.098

Table 6.9: *Average time (in seconds) per pressure equation for solving 100 frames of the circular domain problem in 2D.*

Size	FAS-MG	PI-PCG	PI-MG	PE-PCG	PE-MG	FMG	Newton	No LCP
32	0.037	0.014	0.036	0.015	0.036	0.076	0.98	0.0074
64	0.25	0.2	0.33	0.2	0.37	1.68	25.38	0.09
128	2.15	2.64	2.82	2.82	4.28	30.24	637.67	1.15
256	16.78	36.96	24.58	37.82	47.38	510.71	NA	14.24

Table 6.10: *Average time (in seconds) per pressure equation for solving 100 frames of the spherical domain problem in 3D.*

Size	FAS-MG	PI-PCG	PI-MG	PE-PCG	PE-MG	FMG	Newton
32	0.016	0.0042	0.032	0.004	0.029	0.017	0.41
64	0.092	0.018	0.15	0.018	0.12	0.1	3.5
128	0.38	0.1	0.7	0.11	0.53	1.21	31.35

Table 6.11: *Comparison of the average time (in seconds) between our solvers and Newton's method for solving the pressure equations for 100 frames of the maze problem in 2D.*

Our methods are also more scalable than FMG and Newton's method in terms of timing. When the problem size is doubled, FAS-MG, PI-MG, and PE-MG take about 5 times longer in 2D and 10 times longer in 3D. PI-PCG and PE-PCG's average time increases by about 7 times in 2D and 14 times in 3D when the problem sizes doubles. However, the average time for FMG increases by about 7 times in 2D and 17 times in 3D when the problem size doubles. Newton's method is the worst: about 10 times for 2D and 25 times for 3D.

We briefly discuss the time complexity per iteration for each method. For the multigrid methods (namely FAS-MG, FMG, and our basic MG for linear systems) each smoothing step takes about the same time on the finest grid. Each PCG iteration has about the same time complexity as one smoothing step on the finest grid in multigrid. Due to our Galerkin construction of the coarse grid matrix, our multigrid methods take more time on the coarse grid for smoothing compared to a direct discretization approach, since the coarse grid matrices have more nonzeros per row. However, this deficiency is outweighed by the good scalability of our resulting method. The size of the linear systems in each outer iteration of policy iteration, penalty method, and Newton's method are about the same. However, our policy iteration and penalty method do not need to perform line searches. Updating the policy (in PI) and adding penalty terms (in PE) are both relatively cheap.

6.3.3 Convergence

To demonstrate the convergence behavior of the multigrid solvers (FAS-MG, PI-MG, PE-MG and FMG), we choose specific frames in three different scenarios in 3D defined in section 6.2 such that they have significant handling of fluid solid separation and demonstrate the effects of using separating solid boundary conditions. Specifically, we choose the 10th frame for scenario 1, the 70th frame for scenario 2, the 10th frame for scenario 3. We present the convergence plot in terms of infinity norm of residual against the number of iterations for the three frames in Figures 6.9, 6.10, 6.11, respectively. For FAS-MG, PI-MG, and PE-MG, the number of iteration is the total number of V-cycles. For FMG, it refers to the total number of full cycles.

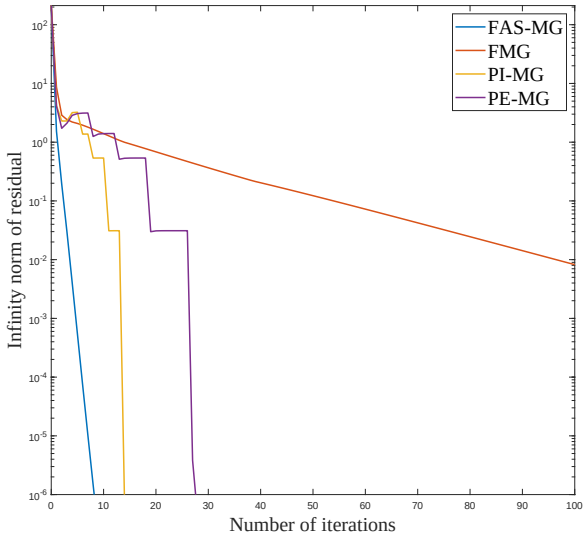


Figure 6.9: *Convergence plots for our methods vs. FMG on a grid of size 256 for the 10th frame in scenarios 1.*

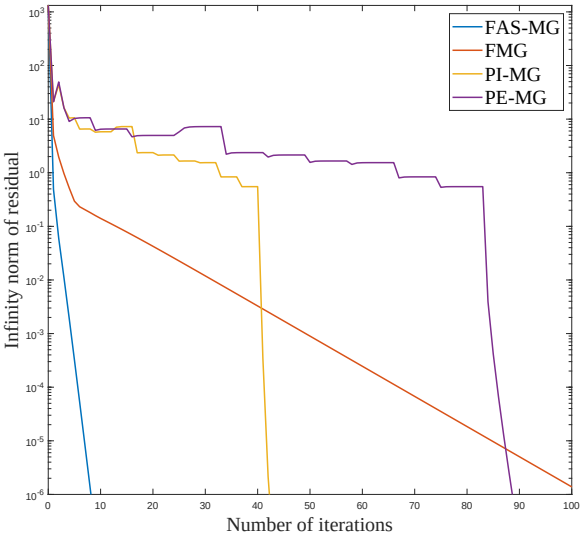


Figure 6.10: *Convergence plots for our methods vs. FMG on a grid of size 256 for the 70th frame in scenarios 2.*

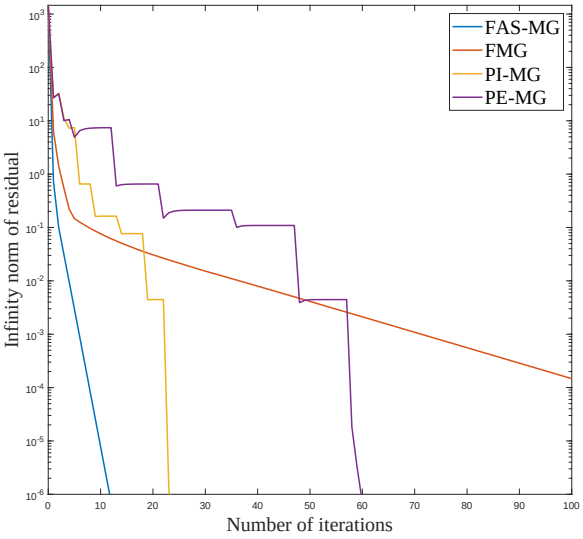


Figure 6.11: *Convergence plots for our methods vs. FMG on a grid of size 256 for the 10th frame in scenario 3.*

For the 10th frame in scenario 1 shown in Figure 6.9, the FAS-MG, the policy iteration, and the penalty method, have similar convergence behavior and are much better than the FMG. For the 70th frame in scenario 2 shown in Figure 6.10, the FAS-MG has similar convergence but the policy iteration and the penalty method become worse. They perform worse than the FMG at the beginning but eventually converge faster than FMG. For the 10th frame in scenario 3 shown in Figure 6.11, the convergence result is similar to scenario 2 except that the policy iteration and the penalty method have better convergence. From the convergence plots, it is clear that the FAS-MG has much better convergence than the others.

6.4 Comparison with the standard FAS-MG

Finally, we show how each of our proposed modifications (interpolation and restriction, boundary handling, and coarse grid matrix construction) on the standard FAS-MG contributes to the success of our FAS multigrid solver. We replace each of them with the simpler or standard option. Specifically, we replace barycentric interpolation and restriction with standard trilinear, or with piecewise constant (PWC) interpolation and restriction; replace the boundary handling introduced in section 5.3.3 with no specialized boundary handling, and replace the Galerkin method for constructing the coarse grid matrix with direct discretization of coarse levels, respectively.

Size	Ours	Trilinear interp.	PWC interp.	Simple boundaries	Direct discretization
32	7	6	47	6	65
64	8	8	112	32	171
128	8	8	268	Diverged	Diverged
256	9	9	616	Diverged	Diverged

Table 6.12: *Number of iterations for solving the pressure using variants of our FAS-MG scheme, at frame 10 of scenario 1 in 3D.*

We present the number of iterations (Table 6.12) on solving the pressure equation for the first substep at the 10th frame of scenario 1 in 3D. Our method with the barycentric interpolation has similar scalability as the standard trilinear interpolation but is more than two times faster. This is expected as the coarse grid matrix obtained from barycentric is about half sparser than from the standard trilinear. As shown in Figure 6.12, the improvement on timing is significant when the problem size is large (128^3 and 256^3). Compared

with the barycentric/trilinear interpolation with higher order of accuracy, the piecewise constant interpolation makes the FAS-MG no longer scalable as the number of iterations doubles with problem size. Table 6.12 also shows that the FAS-MG diverges when using either simple boundary handling or direct discretization for large problems (128^3 and 256^3). This means the special boundary handling and Galerkin method for coarse grid matrix construction are necessary for the convergence of FAS-MG.

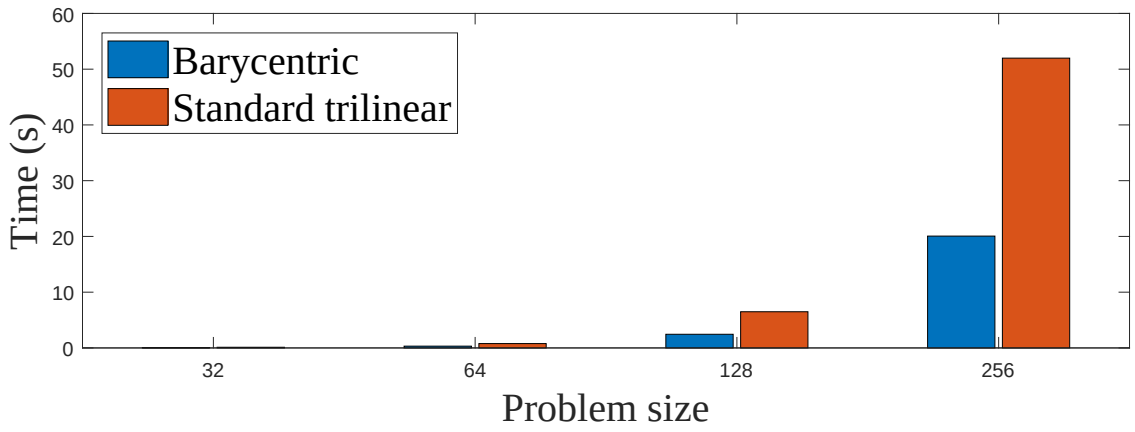


Figure 6.12: *Timing comparison between using barycentric and standard trilinear interpolations for solving the pressure at the 10th frame of scenario 1 in 3D*

Chapter 7

Conclusion and future work

In summary, we have proposed three methods, namely policy iteration, penalty method, and FAS multigrid, as fast solvers for the pressure equations arising from liquid simulation with separating solid boundary conditions. For our FAS multigrid methodology, we introduced several adaptations to achieve the desired mesh-independent convergence behavior on our LCP fluid problem. We demonstrated the superior efficiency and scalability of our resulting solvers over existing methods. Moreover, our results show that our solvers are able to resolve the liquid sticking issue near the solid boundary without making a major sacrifice in computation time compared with the simpler linear solver case.

7.1 Future work

Simulating rigid bodies is a fundamental topic in computer graphics. The main components of rigid body simulation are: time integration, collision detection, and impact response, which is the most time consuming part. After detecting collision of objects, the simulator handles the impact of multiple objects through an impact operator, which determines the post-impact velocity of each object. The main challenges are developing a correct impact operator so that the objects behave as desired after collision; and solving the post-impact velocities efficiently.

Smith et al. [53] propose a generalized reflections multi-impact operator (known as GR operator), which combines the LCP with Gauss-Seidel formulations, to satisfies all five desiderata in rigid body simulations. The nonlinear optimization package Ipopt [59] is used to solve the LCP but the scalability is not clear yet in terms of timing from their

performance tests. Enzenhofer et al. [22] compares four mixed linear complementarity problem (MLCP) solvers: Block Principal Pivoting (BPP) [43], Projected Gauss-Seidel (PGS) [23], Projected Gauss-Seidel with Subspace Minimization (PGS-SM) [52], and the Spook Stepper (SPOOK) [45] for multibody simulations with contact. BPP and PGS-SM are more accurate at the cost of computational time. PGS is suitable for simulations where the fast computation is the highest priority and accuracy is secondary. SPOOK works well only for cases where the friction forces are not decisive.

The existing algorithms in collision response modeling cannot model multi-impact in rigid body simulation realistically as they fail to fulfill at least one of five physical desiderata, which are Break away (BRK), Symmetry preserved (SYM), Energy bounded (KIN), Momentum conserved (MOM), and One-sided impulses (ONE). BRK means bodies previously in contact may break away from each other due to the impact. SYM is that spatial symmetries from pre-impact configurations should also exist in post-impact ones. KIN makes sure that kenetic energy does not increase. MOM conserves the total momentum. ONE means that impulses may push bodies apart but not pull them together. The GR impact operator resolves the issues from the existing algorithms by satisfying all of the five physical desiderata.

Our goal is to speed up the GR impact operator by replacing the LCP solver inside its loop with faster solvers, including policy iteration, penalty method, and multigrid. We describe some potential challenges we may face. The matrix in the LCP, unlike the one from the pressure equation, is not a Poission matrix. Some methods like policy iteration may not converge as the condition for that is no longer satisfied in the problem. We actually find that most matrices are even singular. The solution is also sensitive to the modelling, which means the LCP constraints must be strictly satisfied otherwise the simulation may go wrong. In such cases a solver may not give the correct solution that satisfies the five physical desiderata. We would like to explore the possibility of making changes to the existing rigid body model (GR operator) to get an M-Matrix while preserving the five desiderata or modifying policy iteration, penalty method, and multigrid such that they still work well with those ill-conditioned matrices.

References

[1] Graham. https://docs.computecanada.ca/wiki/Graham, 2017.

[2] Ilan Adler and Sushil Verma. The linear complementarity problem, lemke algorithm, perturbation, and the complexity class ppad. 2011.

[3] Michael Andersen, Sarah Niebe, and Kenny Erleben. A fast linear complementarity problem (lcp) solver for separating fluid-solid wall boundary conditions. In *Proceedings of the 13th Workshop on Virtual Reality Interactions and Physical Simulations*, pages 39–48. Eurographics Association, 2017.

[4] Michael Andersen, Sarah Niebe, and Kenny Erleben. A fast linear complementarity problem solver for fluid animation using high level algebra interfaces for gpu libraries. *Computers & Graphics*, 69:36–48, 2017.

[5] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 23–34. ACM, 1994.

[6] Christopher Batty. A 2d implementation of the sca 2008 paper accurate viscous free surfaces by batty and bridson. https://github.com/christopherbatty/VariationalViscosity2D, 2008.

[7] Christopher Batty. A simple 3d grid-based simulator for animating inviscid free surface liquids in irregular domains. https://github.com/christopherbatty/Fluid3D, 2011.

[8] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. In *ACM Transactions on Graphics (TOG)*, volume 26, page 100. ACM, 2007.

[9] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[10] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

[11] Robert Bridson. Fluid simulation for computer graphics. ak peters series. *Taylor & Francis*, 3:10, 2008.

[12] William L Briggs, Steve F McCormick, et al. *A multigrid tutorial*, volume 72. Siam, 2000.

[13] Yangang Chen. Numerical methods for hamilton-jacobi-bellman equations with applications. 2019.

[14] Nuttapong Chentanez and Matthias Mueller-Fischer. A multigrid fluid pressure solver handling separating solid boundary conditions. *IEEE transactions on visualization and computer graphics*, 18(8):1191–1201, 2012.

[15] Nuttapong Chentanez and Matthias Müller. Real-time eulerian water simulation using a restricted tall cell grid. In *ACM Transactions on Graphics (TOG)*, volume 30, page 82. ACM, 2011.

[16] Richard W Cottle, Jong-Shi Pang, and Richard E Stone. The linear complementarity problem. academic pr ess. *Inc., Boston, MA*, 1992.

[17] Hadrien Courtecuisse and Jérémie Allard. Parallel dense gauss-seidel algorithm on many-core processors. In *2009 11th IEEE International Conference on High Performance Computing and Communications*, pages 139–147. IEEE, 2009.

[18] Christian Dick, Marcus Rogowsky, and Rüdiger Westermann. Solving the fluid pressure poisson equation using multigrid—evaluation and improvements. *IEEE transactions on visualization and computer graphics*, 22(11):2480–2492, 2015.

[19] Zdenek Dostál and Joachim Schoberl. Minimizing quadratic functions subject to bound constraints with the rate of convergence and finite termination. *Computational Optimization and Applications*, 30(1):23–43, 2005.

[20] Yann d'Halluin, Peter A Forsyth, and George Labahn. A penalty method for american options with jump diffusion processes. *Numerische Mathematik*, 97(2):321–352, 2004.

[21] Stanley C Eisenstat. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM Journal on Scientific and Statistical Computing*, 2(1):1–4, 1981.

[22] Andreas Enzenhöfer, Sheldon Andrews, Marek Teichmann, and József Kövecses. Comparison of mixed linear complementarity problem solvers for multibody simulations with contact. In *Proceedings of the 14th Workshop on Virtual Reality Interactions and Physical Simulations*, pages 11–20. Eurographics Association, 2018.

[23] Kenny Erleben. Velocity-based shock propagation for multibody dynamics animation. *ACM Transactions on Graphics (TOG)*, 26(2):12, 2007.

[24] Kenny Erleben. Open source project for numerical methods for linear complementarity problems in physics-based animation. https://github.com/erleben/num4lcp, 2011.

[25] Kenny Erleben. Numerical methods for linear complementarity problems in physics-based animation. In *Acm Siggraph 2013 Courses*, page 8. ACM, 2013.

[26] Michael C Ferris and Todd S Munson. Complementarity problems in gams and the path solver. *Journal of Economic Dynamics and Control*, 24(2):165–188, 2000.

[27] Florian Ferstl, Rüdiger Westermann, and Christian Dick. Large-scale liquid simulation on adaptive hexahedral grids. *IEEE transactions on visualization and computer graphics*, 20(10):1405–1417, 2014.

[28] PA Forsyth. Sparseit++ user guide. *Department of Computer Science, University of Waterloo, august*, 20, 2010.

[29] Peter A Forsyth and Hong Jiang. Nonlinear iteration methods for high speed laminar compressible navier-stokes equations. *Computers & fluids*, 26(3):249–268, 1997.

[30] Peter A Forsyth and George Labahn. Numerical methods for controlled hamilton-jacobi-bellman pdes in finance. *Journal of Computational Finance*, 11(2):1, 2007.

[31] Peter A Forsyth and Kenneth R Vetzal. Quadratic convergence for valuing american options using a penalty method. *SIAM Journal on Scientific Computing*, 23(6):2095–2122, 2002.

[32] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, 2001.

[33] Jorge Gascón, Javier S Zurdo, and Miguel A Otaduy. Constraint-based simulation of adhesive contact. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 39–44. Eurographics Association, 2010.

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

[35] Michael Griebel, Thomas Dornseifer, and Tilman Neunhoeffer. *Numerical simulation in fluid dynamics: a practical introduction*, volume 3. Siam, 1997.

[36] Eran Guendelman, Andrew Selle, Frank Losasso, and Ronald Fedkiw. Coupling water and smoke to thin deformable and rigid shells. *ACM Transactions on Graphics (TOG)*, 24(3):973–981, 2005.

[37] Dong Han and Justin WL Wan. Multigrid methods for second order hamilton–jacobi–bellman and hamilton–jacobi–bellman–isaacs equations. *SIAM Journal on Scientific Computing*, 35(5):S323–S344, 2013.

[38] Francis H Harlow and J Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.

[39] Ronald A Howard. Dynamic programming and markov processes. 1960.

[40] Tiffany Inglis, M-L Eckert, James Gregson, and Nils Thuerey. Primal-dual optimization for fluids. In *Computer Graphics Forum*, volume 36, pages 354–368. Wiley Online Library, 2017.

[41] H Jiang and PA Forsyth. Robust linear and nonlinear strategies for solution of the transonic euler equations. *Computers & fluids*, 24(7):753–770, 1995.

[42] Hong Jiang and Peter A Forsyth. Robust numerical methods for transonic flows. *International journal for numerical methods in fluids*, 24(5):457–476, 1997.

[43] Joaquim J Júdice and Fernanda M Pires. A block principal pivoting algorithm for large-scale strictly monotone linear complementarity problems. *Computers & operations research*, 21(5):587–596, 1994.

[44] Peter Kipfer. Lcp algorithms for collision detection using cuda. *GPU Gems*, 3:723–740, 2007.

[45] Claude Lacoursière and Mattias Linde. Spook: a variational time-stepping scheme for rigid multibody systems subject to dry frictional contacts. *UMINF report*, 11, 2011.

[46] Liang Li, Ting-Zhu Huang, Yan-Fei Jing, and Zhi-Gang Ren. Effective preconditioning through minimum degree ordering interleaved with incomplete factorization. *Journal of computational and applied mathematics*, 279:225–232, 2015.

[47] John E Lloyd. Fast implementation of lemke's algorithm for rigid body contact simulation. In *Proceedings of the 2005 ieee international conference on robotics and automation*, pages 4538–4543. IEEE, 2005.

[48] Aleka McAdams, Eftychios Sifakis, and Joseph Teran. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 65–74. Eurographics Association, 2010.

[49] 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*, pages 9–18. Eurographics Association, 2008.

[50] Rahul Narain, Abhinav Golas, and Ming C Lin. Free-flowing granular materials with two-way solid coupling. In *ACM Transactions on Graphics (TOG)*, volume 29, page 173. ACM, 2010.

[51] Stanley Osher, Ronald Fedkiw, and K Piechor. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.*, 57(3):B15–B15, 2004.

[52] Morten Silcowitz, Sarah Niebe, and Kenny Erleben. Interactive rigid body dynamics using a projected gauss–seidel subspace minimization method. In *International Conference on Computer Vision, Imaging and Computer Graphics*, pages 218–229. Springer, 2010.

[53] Breannan Smith, Danny M Kaufman, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. Reflections on simultaneous impact. *ACM Transactions on Graphics (TOG)*, 31(4):1–12, 2012.

[54] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999.

[55] Tetsuya Takahashi and Christopher Batty. Monolith: a monolithic pressure-viscosity-contact solver for strong two-way rigid-rigid rigid-fluid coupling. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

[56] Jie Tan and XuBo Yang. Physically-based fluid animation: A survey. *Science in China Series F: Information Sciences*, 52(5):723–740, 2009.

[57] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000.

[58] RS Varga. Matrix iterative analysis second. *Springer Series in Computational Mathematics*, 27, 2009.

[59] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[60] Justin WL Wan and Xu-Dong Liu. A boundary condition–capturing multigrid approach to irregular boundary problems. *SIAM Journal on Scientific Computing*, 25(6):1982–2003, 2004.

[61] Katsu Yamane and Yoshihiko Nakamura. A numerically robust lcp solver for simulating articulated rigid bodies in contact. *Proceedings of robotics: science and systems IV, Zurich, Switzerland*, 19:20, 2008.

APPENDICES

Appendix A

Proof of M-Matrix

We give a proof that the matrix $\mathbf{A}$ in (3.41) is an M-Matrix. The matrix $\mathbf{A}$ is an M-Matrix [13] if it has the following properties:

- L-Matrix: $A_{i,i} > 0$ for all i and $A_{i,j} \leq 0$ for all $i \neq j$;
- Diagonally dominant: $|A_{i,i}| \geq \sum_{j \neq i} |A_{i,j}|$ for all i;
- Connectivity: $\mathscr{G}(\mathbf{A}) = \{i | |A_{i,i}| > \sum_{j \neq i} |A_{i,j}|\} \neq \emptyset$, and for any $i \notin \mathscr{G}(\mathbf{A})$, there exists a sequence $i_0, i_1, ..., i_k$ with $A_{i_r, i_{r+1}} \neq 0$, $0 \leq r \leq k-1$ such that $i_0 = i$ and $i_k \in \mathscr{G}(\mathbf{A})$.

We have shown in the discretization that $\mathbf{A}$ satisfies the first two conditions: L-Matrix and diagonally dominant. Now we explain how it satisfies the connectivity when there is air in the domain. As we mentioned earlier, for the cell near the fluid-air surface, its corresponding row is strictly diagonal dominant, which means $\mathscr{G}(\mathbf{A})$ is not empty. We call two cells corresponding to row i and j of $\mathbf{A}$ are connected if $A_{i,j} \neq 0$. This happens when the fluid covers their shared edges/faces. Therefore, to satisfy the connectivity property for an index i, there must exist a path from the corresponding cell to a cell near the fluid-air surface. We can prove such path exists by contradiction. Assume it does not exist for a cell, then any path initiating from that cell will only lead to solid or fluid instead of air, this contradicts the assumption that there is air in the domain.