A Scalable Method for Many Object Fluid-Structure Interaction Simulations

by

Connor Tannahill

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

© Connor Tannahill 2021

Authors 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

Fluid-Structure Interaction (FSI) Simulations are an important technology in many areas of research, including, but not limited to, Computer Graphics, Computational Physics and Engineering. This area is concerned with the realistic simulation of fluids, solids and their interaction. To accurately realize a simulation of an FSI scenario typically requires a large amount of computation and specialized numerical methods to ensure stability and accuracy. In this thesis, we consider the development of a numerical method for dealing with two and three-dimensional FSI scenarios where a large number of deformable objects are immersed in a incompressible fluid. The fluid component of the model is solved using standard CFD approaches, while the solid models are computed using an efficient heuristic. Their interaction is coordinated through the use of a specialized algorithm based on the level set method to coordinate the fluid solver and the solid models, transfer the necessary information between these model components, as well as object-with-object collisions, using a novel collision handling algorithm for arbitrarily shaped deformable objects.

The method is described along with the motivations behind each model component in how they help us reach the goal of a scalable FSI object for this general scenario. Several test cases are presented to demonstrate the capability of the method in producing realistic FSI simulations. These experiments are then analysed to establish the scaling performance of our solver in terms of relevant performance metrics.

Acknowledgements

Firstly I want to thank my family, because I'm pretty sure that's what you're supposed to do. Love ya.

Secondly, a thank you to my supervisor Justin Wan. Your support and guidance during this strange time for research was instrumental. I'm looking forward to continuing our work together in the years to come.

A special thank you to my road biking crew and friends in the program, Matthew Rafuse and Thomas Humphries. I hope the old saying is true: "friends who suffer together pointlessly for hundreds of kilometres, stay together pointlessly for hundreds of kilometres"

And to my friend Amelia, who wrote a way better acknowledgement for me in her thesis than I could come up with. You win this round.

Also a thank you to my readers Christopher Batty and Yuying Li. Your excellent comments and questions on my thesis were greatly appreciated.

Dedication

You're so vain (you probably think this thesis is about you).

Table of Contents

List of Tables viii

List of Figures ix

1 Introduction 1

2 Previous Work 3

3 Physical Theory 6

3.1 Physical Theory for Fluids 6

3.2 Physical Theory for Solids 9

4 Fluid and Solid Simulation Methods 13

4.1 Numerical Method for Fluids 13

4.1.1 Explicit-Implicit Solution Procedure for the Navier-Stokes equation 13

4.1.2 Spatial Discretization 16

4.1.3 Boundary Conditions 20

4.2 Interface Representation 23

4.2.1 Interface Motion 26

4.2.2 The Fast Marching Method 26

4.3 Deformable Solid Simulation 30

4.3.1 Approximation Using Mass-Spring Assemblies 30

4.3.2 Modelling Mass-Spring Systems 34
4.3.3 Integrating Mass-Spring Systems 36

5 FSI Procedure **40**
5.1 Fluid-Structure Interaction 40
5.1.1 Communication between the MSS's and the Implicit Surface Representation 42
5.2 Collision Handling 45
5.3 Summary of Simulation Algorithm 50

6 Results **54**
6.1 Scenario 1: Lid Driven Cavity 54
6.2 Scenario 2: Object Tumbling 56
6.3 Scenario 3: Flow Through Channel 58
6.4 Scaling Performance 59
6.5 Three-dimensional Results 62

7 Conclusions and Future Work **78**

References **79**

List of Tables

6.1 Object scale test data for each of the test scenarios. 62
6.2 Full scaling data for each of the test scenarios 63
6.3 Full scaling data for each of the test Scenarios in 3D 65
6.4 Object scaling data for Scenario 1 in 3D, $\boldsymbol{N} = (320, 320, 320)$ run to $t_{\text{end}} = 1.5$ 65

List of Figures

4.1 2D Staggered Grid 17

4.2 3D Staggered Grid Cell 18

4.3 2D Stencil for the velocity terms in the momentum equation, centered at the x-component of the fluid velocity 19

4.4 2D Ghost Cells 21

4.5 Grid labelling of grid cells as solid and fluid based on value of signed distance function for a circle. 24

4.6 Two different representations of a circle using the level set method. First row is a circle attained using the level set function corresponding to the standard equation of a circle $\phi(x, y) = x^2 + y^2 - 0.5^2$. Second row is attained using the level set function corresponding to the equation of a cone $\phi(x, y) = \sqrt{x^2 + y^2} - 0.5$, the signed distance representation of a circle. 25

4.7 Mass-Spring System assembly for a circle and a sphere. 31

4.8 (Left) The components of the two-dimensional Cauchy strain tensor in terms of how they affect the deformation of a solid volume. (Right) The analogous two-dimensional MSS, where superscripts indicate some that two edges may have duplicate roles in terms of the analogy between the two.. 32

4.9 Visualization of the effect of the three-dimensional Cauchy Stress tensor on a volume. 33

4.10 Spring and Dashpot Linear Element Diagrams. 33

4.11 Maxwell and Kelvin Model Diagrams. 34

5.1 Determining the solid wall boundary condition on the staggered grid. Three dimensional case works analogously. 42

5.2 Extrapolated velocity and speed fields for a circle undergoing deformation. 44

5.3 (Left) Collision detection process using radius search from medial axis points of the current object boundaries, Φ_t. (Right) Computing the spring-based repulsive force based on the proximity of a node on object A to nodes in object B at the time of collision detection. 47

5.4 Graphical representation of regions of the domain identified as being closest to a given object, as labelled during our modified FMM. 48

5.5 Basic collision between 2 convex objects in absence of fluid (initial object velocities chosen by hand). The vectors are the normally extrapolated velocities from the surface of the objects. The objects proceed towards each other until a collision is detected, the collision handling them applies the collision forces. This results in deformation in each of the objects, and for them to bounce away from each other. Note that some distance δ_{col} must be maintained between objects, hence the large gaps for coarse grids. . . . 50

5.6 Collision between convex objects moving at varying initial velocities to induce collisions. We see here the capabilities of the collision detection algorithm for managing collisions between several objects simultaneously. . . . 51

5.7 Basic collision between a convex and concave objects as a demonstration of our ability to handle non-convex object collision. 52

6.1 Scenario 1 diagram. 55

6.2 Scenario 1 realization for varying numbers of objects and grid size. (Top) 10 seeded circular objects on a grid of size $\boldsymbol{N} = (80, 80)$. (Bottom) 30 seeded circular objects on a grid of size $\boldsymbol{N} = (160, 160)$. in each case the solution is given at $t = 5, 10, 15$. 56

6.3 Tumbling Scenario for 10 falling circular objects over time $\boldsymbol{N} = (80, 160)$ running to $t = 10$. 66

6.4 Scenario 2 with Dense (118) Falling Objects with grid size $\boldsymbol{N} = (320, 640)$ running to $t = 10$. 67

6.5 Close-up of initial collision point in Figure 6.4 at times $t = 0.5, 1, 1.5$ (from left-to-right). 68

6.6 Scenario 3 diagram. 68

6.7 Scenario 3 realization. $\boldsymbol{N} = (320, 160)$ with 40 immersed objects. Pictured at instances $t = 3.35, 6.65, 10$ (top to bottom). 69

6.8 Scenario 3 realization. $\boldsymbol{N} = (320, 160)$ with 700 immersed objects. Pictured at instances $t = 3.35, 6.65, 10$ (top to bottom). 70

6.9 Visualization of the minimum object size for a simulation with grid cell size h. 71

6.10 Scenario 1 - Plot of the total (left) and average per-step (right) CPU time to reach t_{end}. 71

6.11 Scenario 2 - Plot of the total (left) and average per-step (right) CPU time to reach t_{end}. 72

6.12 Scenario 3 - Plot of the total (left) and average per-step (right) CPU time to reach t_{end}. 72

6.13 Scenario 2 Basic Example with a grid of size $\boldsymbol{N} = (160, 160, 160)$ and 49 immersed objects at $t = 5$ (top) and $t = 10$ (bottom) 73

6.14 Scenario 2 Basic Example with Course Grid and 49 immersed objects at $t = 15$ (top) and $t = 20$ (bottom) 74

6.15 Scenario 3 Basic Example with Course Grid and 49 immersed objects at $t = 2.5$ (top) and $t = 5$ (bottom) 75

6.16 Scenario 3 Basic Example with Course Grid and 49 immersed objects at $t = 7.5$ (top) and $t = 10$ (bottom) (Give more details on each of the scenarios in-text) 76

6.17 Simultaneous scaling results for Scenario Two in 3D. 77

Chapter 1

Introduction

Fluid-Structure Interaction (FSI) is a classical field of scientific computing that deals with the problem of accurately simulating the physical interactions between fluids and solid objects [23]. This process integrates the methods of Computational Fluid Dynamics (CFD) for modelling the fluid behaviour, the simulation of often complicated non-linear solid models, as well as specialized methods for addressing the interaction and communication between these model components. FSI simulations are of interest in many areas of study, including computer graphics [2], where FSI simulations can be used to depict visually and physically realistic scenes. Engineering applications use FSI in the design process to analyze topics such as manufacturing processes and the effects of hydrodynamic stresses on products [18]. While the utility of FSI is clear, in general, the interaction between the solid and fluid components of the models is highly non-linear and can be difficult to accurately capture using numerical methods. In particular, achieving high performance while preserving the accuracy and stability of the method can pose a substantial problem.

In this thesis, we consider the case of simulation scenarios where multiple deformable objects are immersed in a viscous, incompressible fluid. An example of a target application for this kind of work is in the simulation of red blood cells being propelled through an artery. This application has been appearing recently in the literature, with possible applications in the study of blood platelet motility for better understanding the dynamics of clot formation, among other things [24, 52]. To perform this kind of FSI simulation, trade off's must be made in terms of accuracy and performance, as performing engineering-quality FSI simulation would be prohibitively expensive.

Many of the existing methods for this problem class make use of particle-based methods for fluid simulation to achieve adequate run time performance [24]. Additionally, heuristic

solid models are used, as performing higher-quality solid deformation computation for each of the many objects, such as using the standard Finite Element Method (FEM), would be an insurmountable performance bottleneck. In our simulation, we take the view standard CFD-based fluid simulation techniques are sufficient for efficiently performing the fluid simulation so long as the fluid-solid and solid-solid interactions are handled in an appropriate way for the problem class in question. Much of our work draws upon existing standard methods for fluid simulation, with modifications to address the performance bottlenecks that exist for this problem class.

The goal of this thesis is the development of an efficient FSI simulation framework for the case of many immersed objects. In particular, we consider how standard CFD methods are frequently seen in the literature can be applied in such a way to facilitate high-performance resolution of interactions at the fluid-solid and solid-solid levels. This results in a simulation that could apply to high-performance CFD tasks, such as the blood cell simulation mentioned previously.

The thesis is structured as follows. In Chapter 2, the physical theory for our fluid and deformable solid models is presented. Chapter 3 then discusses the background methods we make use of in our thesis, particularly the numerical methods used in the fluid simulation and the heuristic solid model we implement to model deformations. Chapter 4 discusses the methodology we use to perform our simulation, describing the motivation and implementation of the component algorithms. Verification of the efficacy of our simulation methods and some preliminary simulations are presented in Chapter 5. Chapter 6 finished with some conclusions, as well as some directions and suggestions for future work.

Chapter 2

Previous Work

Owing to its highly practical and multidisciplinary nature, the literature of FSI as a field is vast. The classical theory of fluid dynamics simulation with physical theories for solid object behaviour. In particular, the kinds of previous work which most align with the goals of this thesis are those appearing in computer graphics, [7] where the interaction of fluids and solids must be rapidly solved to meet high-performance demands in this application. Another related area is in engineering, in which accurate FSI simulation has become a cornerstone in the design stage of many application areas [41]. Further, recent work [24, 52] in the multi-scale simulation of biological phenomena has seen interest for applications such as the simulation of blood cell and platelet motility through blood vessels, which we considered to be the problem area of ultimate interest for this research, though the applications can, of course, be broader than this.

To discuss the literature of the computer simulation of fluids is to discuss a century of developments in various theoretical, numerical and methodological advancements. For this reason, we focus particularly on some of the more recent works which are most relevant to our simulation. For a more general treatment of numerical methods for fluids, see [7] and [33] for an introduction to the area, [7] for a more focused introduction for the application of computer graphics. We refer the readers to [3], [32], and [25] for some recent approaches and developments. Fundamental to our fluid simulation are the methods first purposed in the Marker-and-cell (MAC) methods of Harlow and Welch [20], which introduced the staggered grid arrangement which we use in all of our fluid-based numerical methods. This has become a standard method due to its good accuracy and stability properties.

Due to its many difficulties, FSI itself is a large topic of study. For an excellent recent review of FSI as a field see [23]. The most typical classification of different FSI approaches

is into monolithic and partitioned schemes. In monolithic schemes, the fluid and solid components of the FSI models are considered to evolve simultaneously. This typically involves some kind of iteration at the fluid-solid interface in order to bring the two model components to convergence.

The classical example of a monolithic approach is the immersed boundary method of Peskin [39], which makes use of an implicit boundary representation on a uniform grid for simplicity and applies iteration at grid cells on the solid-fluid boundary to achieve convergence by filling the solid region with artificial fluid and achieving convergence to a variational formulation of the equations [40]. The advantage of the monolithic approach is that it may achieve better accuracy in general, and the interface of the two model components is done implicitly, which avoids the requirement that the interface is closely tracked.

Partitioned schemes for FSI consider the fluid and solid model components to be governed by two separate solvers. Each model component is solved independently and then coupled together at the end. The primary disadvantage of a partitioned approach is first that the accuracy and stability of the method can be harder to guarantee, and that coordinating the different aspects of the model may be difficult at times. Common partitioned approaches usually make use of a conforming mesh in order to give accurate resolution on the fluid-solid interface so that the boundary conditions will be well-captured [23]. This is frequently done using the Arbitrary Lagrangian-Eulerian method, where the mesh nodes are refined close to the boundary of the object to satisfy certain geometric conservation laws [10]. In our work, we instead consider a partitioned approach, in which the fluid and solid model components are governed by two separate solvers, between which information is communicated. The advantage of this approach is that separate numerical solvers can be used for each component, allowing for more flexibility.

Similar solvers to ours include the solver introduced by [15] for the simulation of deformable solids in computer graphics simulation. Here the MAC method is extended through the introduction of grid cells representing springs connected along cardinal directions to neighbouring spring cells. These cells impose spring forces onto the neighbouring fluid cells, and a volume-of-fluid [47] based method is used to determine whether a cell belongs to the object. These methods use voxelized three-dimensional grids to provide a framework for fluid simulation which is extensible to many physical scenarios, including two-phase flows [45] and FSI [15]. This includes many specialized methods and improvements, one such example is the work of [2], which replaces the standard projection step with a minimization of the kinetic energy of the system, resulting in more accurate and robust solutions on grids.

As mentioned previously, when developing this thesis our target application is the multi-scale simulation of blood flow with many interacting red blood cells. Recently some such solvers have begun to appear in the literature. The fully Eulerian method [21] based on the Volume of Fluid formulation for an incompressible fluid and Neo-Hookean materials which boasts impressive parallel performance by eliminating the typical pressure Poisson equation solve with an artificial compressibility method which has reasonable guarantees the fluid will be nearly incompressibility. The solver Hemocell [52] makes use of a particle-based fluid simulator (see [34, 25] for reviews on these particle methods) in conjunction with Mass-Spring based solid models in order to model the deformation and motion of large numbers of red blood cells through a capillary. This model is the most similar to ours methodologically, however it does not handle the phenomena of object collision, justifying this choice due to lubrication effects which in general will prevent object collisions. The most recent work in this area is another particle-based method [24] based on the lattice method for fluid simulation, and a nodal finite element method based on a projective dynamics framework of the kind we apply in Section 4. Additionally, Kotsalos et al. [24] does do simple collision avoidance by applying spring forces between colliding objects such that they diverge from each other over time. This is the most complete and well-tested red blood flow simulator we have seen in the literature.

The most related work is the recent paper [35], which simulated many immersed rigid particles interacting within a pool. Rigid body collisions between these objects are performed by first detecting the collision through the use of probes extending from the normal of the object, and then approximating the parameters of the rigid body collision. Methodologically, this approach is very similar to ours, where an implicit surface representation through a signed distance function on a uniform grid is used for collision handling between objects, and the remaining differences in the scheme have to do with the fluid and solid implementations. In particular, as this paper only considers rigid bodies, our method can be thought of as an extension of this work, where we add the consideration of deformable objects.

Chapter 3

Physical Theory

In this section, we describe the basic physical theory that we apply throughout this thesis. We first describe the model of fluid dynamics, which is based on the standard Navier-Stokes equations. Secondly, we introduce some of the basic theory for deformable objects of the kinds considered in this thesis. This is not an exhaustive survey, as both the physical theory for fluids and deformable solids encompass vast amounts of literature, though references are provided for further reading.

3.1 Physical Theory for Fluids

To begin with the theory of our FSI simulation, we first must discuss the physical theory of fluid dynamics. One of the triumphs of fluid dynamics is the derivation of simple governing equations which can be used to derive expressions for fluid behaviour in a vast array of physical scenarios [26]. In our simulation, we consider the case in viscous, incompressible fluids such as water. For fluids of this kind, the physical model we use is the incompressible Navier-Stokes equations. These equations can be derived from simple physical principles, which we will now describe. Our derivation is based on the one presented in the textbook [18].

We consider the spatial domain of our fluid to be an open and bounded d dimensional subdomain $\Omega_0^f \subset \Omega^d \subset \mathbb{R}^d$ where Ω_0^f denotes the fluid domain at the initial time and Ω^d is the entire simulation domain. At each point in time $t \in [0, t_{\text{end}}]$ the fluid flow is characterized by the velocity field $\boldsymbol{u} : \Omega_t^f \mapsto \mathbb{R}^d$, a pressure field $p : \Omega_t^f \mapsto \mathbb{R}$ and a density field $\rho : \Omega_t^f \mapsto \mathbb{R}$. Here, Ω_t^f is the fluid domain at time t. Note that in the case of

incompressible flows, we consider the differences in density to be negligible and hence we take ρ to be constant.

To begin the derivation, we define the function $\mathbf{\Phi} : \Omega_0^f \mapsto \Omega_t^f$ which tracks the motion of fluid particle positions $\boldsymbol{c} \in \Omega_0^f$, from its initial configuration to its current configuration. That is, at time t the fluid occupies the transformed domain $\Omega_t^f = \{\mathbf{\Phi}(\boldsymbol{c}, t) : \Omega_0^f\}$. The path of the particle $\boldsymbol{c}$ is traced by the graph of $\mathbf{\Phi}$ as t increases, and the velocity of this particle is given by

$$\boldsymbol{u}(\boldsymbol{x}, t) = \frac{\partial}{\partial t}\mathbf{\Phi}(\boldsymbol{c}, t). \tag{3.1}$$

The first physical law we consider is conservation of mass. Note that our definition of $\mathbf{\Phi}$ means that this is a closed system, as the motion of the particles is such that they are contained in a bounded domain at each point in time. The amount of fluid mass in a region is the integral of the density over this region, and therefore conservation of mass is represented mathematically as

$$\int_{\Omega_0^f} \rho(\boldsymbol{x}, 0) d\boldsymbol{x} = \int_{\Omega_t^f} \rho(\boldsymbol{x}, t) d\boldsymbol{x}. \tag{3.2}$$

Differentiating (3.2) leads to

$$\frac{d}{dt}\int_{\Omega_t^f} \rho(\boldsymbol{x}, t) d\boldsymbol{x} = 0. \tag{3.3}$$

From the Reynolds transport theorem, we have that

$$\frac{d}{dt}\int_{\Omega_t^f} \rho(\boldsymbol{x}, t) d\boldsymbol{x} = \int_{\Omega_t^f} \frac{\partial}{\partial t}\rho(\boldsymbol{x}, t) + \nabla \cdot (\rho\boldsymbol{u})(\boldsymbol{x}, t) d\boldsymbol{x}. \tag{3.4}$$

Since (3.4) holds for any arbitrary Ω_t^f, this implies that

$$\int_{\Omega_t^f} \frac{\partial}{\partial t}\rho(\boldsymbol{x}, t) + \nabla \cdot (\rho\boldsymbol{u})(\boldsymbol{x}, t) d\boldsymbol{x} = 0, \tag{3.5}$$

which implies that

$$\frac{\partial}{\partial t}\rho(\boldsymbol{x}, t) + \nabla \cdot (\rho\boldsymbol{u})(\boldsymbol{x}, t) d\boldsymbol{x} = 0. \tag{3.6}$$

For the case of an incompressible fluid, $\rho = \rho_\infty$ is constant in time and space and therefore we obtain the *continuity equation*

$$\nabla \cdot \boldsymbol{u} = 0. \tag{3.7}$$

The second physical law in the derivation of the Navier-Stokes equations is conservation of momentum. The momentum of all the particles in the domain is written as

$$\boldsymbol{m}(\boldsymbol{x}, t) = \int_{\Omega_t^f} \rho(\boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{x}, t) d\boldsymbol{x}. \tag{3.8}$$

We must now determine how the linear momentum changes with time. The forces acting on a fluid particle at any point in time are broken into two kinds, *body forces* such as gravity and magnetic force, and *surface forces* such as pressure and internal friction. Body forces are represented by

$$\int_{\Omega_t^f} \rho \boldsymbol{g} d\boldsymbol{x}, \tag{3.9}$$

while surface forces are represented by

$$\int_{\partial\Omega_t^f} \rho(\boldsymbol{\tau} \cdot \boldsymbol{n}) d\boldsymbol{x}, \tag{3.10}$$

where $\partial\Omega_t^f$ is the boundary of region Ω_t^f, $\boldsymbol{\tau} \in \mathbb{R}^{d\times d}$ is the hydrodynamic stress tensor, $\boldsymbol{g}$ is a vector field representing the body forces, and $\boldsymbol{n}$ is the outward normal vector. Then by Newton's second law we have

$$\int_{\Omega_t^f} \rho(\boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{x}, t) d\boldsymbol{x} = \int_{\Omega_t^f} \rho \boldsymbol{g} d\boldsymbol{x} + \int_{\partial\Omega_t^f} \rho(\boldsymbol{\tau} \cdot \boldsymbol{n}) d\boldsymbol{x}, \tag{3.11}$$

which, by applying the transport product rule and transport theorem on the left and the divergence theorem to the term on the right results in the *momentum equation*

$$\frac{\partial}{\partial t}(\rho \boldsymbol{u}) + (\boldsymbol{u} \cdot \nabla)(\rho \boldsymbol{u}) + (\rho \boldsymbol{u})(\nabla \cdot \boldsymbol{u}) - \rho \boldsymbol{g} - \nabla \cdot \boldsymbol{\tau} = 0. \tag{3.12}$$

The definition of the stress tensor $\boldsymbol{\tau}$ determines the behaviour of the fluid. In the case of Newtonian Fluids obeying the Stokes Assumption [18] define $\boldsymbol{\tau}$ as the sum of a pressure tensor $-p\boldsymbol{I}$ and a viscosity tensor $\boldsymbol{\tau}_{\text{visc}}$,

$$\boldsymbol{\tau} = -p\boldsymbol{I} + \boldsymbol{\tau}_{\text{visc}} = (-p + \lambda \nabla \cdot \boldsymbol{u})\boldsymbol{I} + 2\mu\boldsymbol{\delta}. \tag{3.13}$$

where μ, λ are material constants and the strain tensor $\boldsymbol{\delta}$ is defined by

$$\boldsymbol{\delta}_{i,j} = \frac{1}{2}\Big(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\Big). \tag{3.14}$$

Substituting into (3.12) and applying the continuity equation (3.7), we have completed the derivation of the divergence form of the Navier-Stokes equations,

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

$$\nabla \cdot \boldsymbol{u} = 0, \tag{3.16}$$

where $\mu = \frac{v}{\rho_\infty}$ is the kinematic viscosity, with v being the dynamic viscosity of the fluid and ϕ_∞ is the constant density of the incompressible fluid.

3.2 Physical Theory for Solids

The physical theory for deformable solid objects has is at its core the study of relations between stress and strain responses. When talking about an object undergoing deformation, stress refers to forces acting on the object, with external stresses referring to external forces which cause the object to undergo deformation, and internal forces referring to stresses due to a change of potential energy in the body of the object. While stress refers to the forces, acting on the object strain refers to the actual deformation of the material itself. The forces are interrelated, with a change in stress resulting in strain on the material, and also a change in strain resulting in a change in potential energy which can result in additional stress within the material. How these response curves are formulated depends greatly on the material in question, and are roughly divided into linear and non-linear materials. Further, materials can be homogeneous and non-homogeneous. In homogeneous materials, the stress-strain response is uniform throughout the continuous body of the object. With non-homogeneous materials, the response may differ depending on the location within the body. While in reality, most objects are non-homogeneous, significant simplifications are made in simulating solid deformations to make the problem computationally tractable, with reasonable correspondence with the actual material behaviour. In this section, we discuss the basic theory of deformable solids, where we draw our information from the texts [11, 4, 14].

Consider a continuous body as a volume $\Omega_t^{s_i} \subset \Omega^d$, where the index i denotes that this body is the *ith* object under consideration. Note that in our simulation, $\Omega_t^f \cup \Omega_t^s =$

Ω^d, where $\Omega_t^s = \cup_i \Omega_t^{s_i}$ is referred to as the structural domain of the simulation. As the simulation proceeds, based on the interaction between the solid regions, the solid object will transition between configurations $\Omega_t^{s_i}$ at time t, with the original configuration denoted as Ω_0^s. Let $\boldsymbol{x}(t_0) \in \Omega_0^{s_i}$ be a material point in its initial configuration. Then after a continuous deformation, this material point will be shifted to a new position $\boldsymbol{x}(t) = \boldsymbol{x}(t_0) + \boldsymbol{d}(\boldsymbol{x}(t_0), t)$ within the new configuration for this object, $\Omega_t^{s_i}$, where $\boldsymbol{d}(\boldsymbol{x}(t_0), t)$ is the displacement function.

Consider a small volume in the neighbourhood around the point $\boldsymbol{x} \in \Omega_t^{s_i}$. Then when the material configuration undergoes a deformation, we can consider it as the cumulative effect of the deformation within all of such volumes. The deformation within this volume is governed by a certain strain tensor, the form of which depends on the material being modelled. The Green-Lagrange strain tensor is defined by

$$E_{i,j} = \frac{1}{2}\Big(\frac{\partial d_i}{\partial x_j} + \frac{\partial d_j}{\partial x_i} + \frac{\partial d_\alpha}{\partial x_i}\frac{\partial d_\alpha}{\partial x_j}\Big), \tag{3.17}$$

where the Einstein notation is used for the last term

$$\frac{\partial d_\alpha}{\partial x_i}\frac{\partial d_\alpha}{\partial x_j} = \sum_\alpha^d \frac{\partial d_\alpha}{\partial x_i}\frac{\partial d_\alpha}{\partial x_j}. \tag{3.18}$$

When the deformation is small, we can consider the quadratic terms (3.18) to be negligible, resulting in the simplified Cauchy's infinitesimal strain tensor

$$\epsilon_{i,j} = \frac{1}{2}\Big(\frac{\partial d_i}{\partial x_j} + \frac{\partial d_j}{\partial x_i}\Big). \tag{3.19}$$

Cauchy's strain tensor represents two kinds of strain: longitudinal strain and shear strain. Longitudinal strain corresponds to the change of length along line elements and forms the main diagonal of the strain tensor. The shear strain corresponds to the change in angle between two perpendicular lines resulting from the deformation and is given by the off-diagonal elements.

The strain tensor can then be divided into volumetric strain, ϵ_v, and deviatoric strain, $\epsilon'_{i,j}$ as

$$\epsilon_{i,j} = \left(\begin{bmatrix} \epsilon_v & 0 & 0 \\ 0 & \epsilon_v & 0 \\ 0 & 0 & \epsilon_v \end{bmatrix} + \begin{bmatrix} \epsilon'_{1,1} & \epsilon_{1,2} & \epsilon_{1,3} \\ \epsilon_{2,1} & \epsilon'_{2,2} & \epsilon_{2,3} \\ \epsilon_{3,1} & \epsilon_{3,2} & \epsilon'_{3,3} \end{bmatrix} \right)_{i,j} \tag{3.20}$$

$$= \epsilon_v \delta_{i,j} + \epsilon'_{i,j}, \tag{3.21}$$

where

$$\epsilon_v = \frac{1}{3}(\epsilon_{1,1} + \epsilon_{2,2} + \epsilon_{3,3}) = \frac{1}{3}\epsilon_{k,k}, \tag{3.22}$$

and $\epsilon'_{1,1}, \epsilon'_{2,2}$, and $\epsilon'_{3,3}$ are deviations from ϵ_v. The volumetric strain ϵ_v describes the expansion or contraction of a material without changing its shape, whereas the deviatoric strain $\epsilon'_{i,j}$ describes the distortion of the material with constant volume.

To deform the unit cube about $\boldsymbol{x} \in \Omega_t^{s_i}$, we model the forces acting upon its surface such as friction and pressure. These forces are described using the stress tensor, σ, which is the same dimension as ϵ, with each component describing a force acting upon the surface of the unit cube.

The stress vector $\boldsymbol{s}$ is the force per unit area acting on a surface with outward normal vector $\mathbf{n}$. The relation between the stress vector $\boldsymbol{s}$ and the strain tensor is $\boldsymbol{s}_i = \sigma_{i,j}\boldsymbol{n}_{i,j}$ by Cauchy's theorem, which states that while the state of stress at a point within a body is defined by all the stress vectors associated with all planes passing through the point, knowing three orthogonal stress vectors is sufficient to determine the remaining stresses via linear transformation. The vector projection of $\boldsymbol{s}$ into the unit normal is called the normal stress, and denoted $\boldsymbol{s}_n$, while the projection into the unit tangent is called the shear stress and is written $\boldsymbol{s}_t$.

Similar to the strain tensor, we can decompose the strain tensor into volumetric and deviatoric stress,

$$\sigma_{i,j} = \sigma_v \delta_{i,j} + \sigma'_{i,j}, \tag{3.23}$$

where the notation is equivalent to the strain case. The volumetric stress σ_v describes tensile or compressive forces, whereas the deviatoric stress $\sigma'_{i,j}$ describes shearing forces.

The mechanical behaviour of any material is governed by physical laws which determine the relationship between stress and strain. The first of these basic laws, Cauchy's equations of motion, which state the conservation of linear momentum within a deformable material. In tensoral notation, the equations of motion are

$$\rho \frac{\partial^2 d_j}{\partial t^2} = G_j + \frac{\partial \sigma_{i,j}}{\partial x_i}. \tag{3.24}$$

In other words, the rate of change of the momentum is equal to the sum of the body forces G_j and the surface forces $\frac{\partial \sigma_{i,j}}{\partial x_i}$.

The second of the basic physical laws are the kinematic equations, which describe the motion of a deformation by relating strain to displacement as governed by the definition of the materials strain tensor (3.19). Assuming infinitesimal deformations the kinematic equations are

$$\sigma_{i,j} = \frac{1}{2}\Big(\frac{\partial d_i}{\partial x_j} + \frac{\partial d_j}{\partial x_i}\Big). \tag{3.25}$$

The final set of equations are the constitutive equations, which describe the characteristics of the material. Given two materials of the same mass and shape, their response to external forces may be completely different, and even further, may depend on other factors such as temperature. In general, modelling the constitutive behaviour of specific real-world materials is very complicated. To avoid these complications, we employ idealized materials, whose constitutive equations only consider the relationship between stress and strain. In general, these have the form

$$\sigma_{i,j} = F(\epsilon_{i,j}). \tag{3.26}$$

The stress and strain response is therefore determined by the integration of these constitutive models over the volume of a continuous solid. We discuss methods for solving the constitutive equations for our particular deformable solid models in Chapter 4.

Chapter 4

Fluid and Solid Simulation Methods

In this chapter, we introduce the numerical methods for each of our model components that are at the core of our simulation techniques. We first describe the numerical methods for fluid dynamics we apply and then go into our representations for the interfaces of the solids and the model we use for solid deformations.

4.1 Numerical Method for Fluids

4.1.1 Explicit-Implicit Solution Procedure for the Navier-Stokes equation

The Navier-Stokes equations given by (3.15) and (3.16) can be written in dimensionless form [18]

$$\frac{\partial}{\partial t}\boldsymbol{u} + \left(\boldsymbol{u}\cdot\nabla\right)\boldsymbol{u} + \nabla p = \frac{1}{Re}\Delta\boldsymbol{u} + \boldsymbol{g}, \tag{4.1}$$

$$\nabla\cdot\boldsymbol{u} = 0. \tag{4.2}$$

$Re > 0$ is Reynolds number, which determines the relative magnitude of viscous and inertial forces, with flows having low Re being dominated by viscous forces and those with high Re dominated by advective forces In three dimensions, we write out the components of $\boldsymbol{u}$ as $\boldsymbol{u} = \begin{bmatrix} u & v & w \end{bmatrix}^T$. To explain the notation, the vector Laplacian $\Delta\boldsymbol{u}$ is defined in

three dimensions as

$$\Delta \boldsymbol{u} = \begin{pmatrix} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \\ \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2} \\ \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2} \end{pmatrix}, \tag{4.3}$$

and the operator $(\boldsymbol{u} \cdot \nabla)\boldsymbol{u}$ is defined as

$$(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = \begin{pmatrix} u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} \\ u\frac{\partial w}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} \end{pmatrix}. \tag{4.4}$$

We define

$$\frac{\partial}{\partial t}\boldsymbol{U}(\boldsymbol{x}, t) = \frac{1}{Re}\Delta \boldsymbol{u} - (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \boldsymbol{g}, \tag{4.5}$$

to separate the terms of the Navier-Stokes equations describing the fluid advection and diffusion in absence of pressure. The Navier-Stokes equations can then be written as the superposition for fluid advection terms and pressure correction terms

$$\frac{\partial}{\partial t}\boldsymbol{u}(\boldsymbol{x}, t) = \frac{\partial}{\partial t}\boldsymbol{U}(\boldsymbol{x}, t) - \nabla p(\boldsymbol{x}, t). \tag{4.6}$$

The implicit-explicit method is then to solve this PDE in two steps. Provisional fluid velocities $\boldsymbol{U}$ are first computed at the next time level in absence of pressure. These velocities are then corrected using pressures defined at the next time level, obtained through the solution of a Poisson equation, such that the continuity equation (4.2) is satisfied. The intermittent velocities are obtained by integrating the PDE (4.5) for a time step, typically using Euler's method or a higher-order Runge-Kutta method. The pressure correction is then computed implicitly.

While high order methods are possible [19], we use first order time stepping to solve (4.6) to the next time level. The reason for this over higher order methods becomes clear when the remainder of our simulation methods are taken into account. Integrating (4.1) with respect to t on the interval $[t_n, t_{n+1}]$, we obtain the expression

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \int_{t_n}^{t_{n+1}} \frac{\partial}{\partial t}\boldsymbol{U} \partial t - \int_{t_n}^{t_{n+1}} \nabla p \partial t. \tag{4.7}$$

Here we use the short-hand notation $\boldsymbol{u}^n = \boldsymbol{u}(\boldsymbol{x}, t_n)$, where the superscript denotes the time step. We will approximate the first integral on the right using the left-endpoint quadrature rule (forward Euler's method) and for the second integral we use the right-endpoint rule (backwards Euler's method) and hence this is an explicit-implicit method. The update rule for u^{n+1} is given by

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \frac{\partial}{\partial t}\boldsymbol{U}^n \Delta t - \nabla p^{n+1} \Delta t, \tag{4.8}$$

where $\Delta t = t_{n+1} - t_n$. Note that (4.8) is an approximation only, but with an abuse of notion we still denote it as $\boldsymbol{u}^{n+1}$.

The value of $\frac{\partial}{\partial t}\boldsymbol{U}^n$ can be obtained by the spatial discretization of (4.5), which we describe in the next section. An important question is how the pressure gradient ∇p^{n+1} can be obtained, as it is defined at the next time level. First we take the divergence of (4.8),

$$\nabla \cdot \boldsymbol{u}^{n+1} = \nabla \cdot \frac{\partial}{\partial t}\boldsymbol{U}^n - \Delta t \Delta p^{n+1}. \tag{4.9}$$

Substituting (4.9) into the continuity equation (4.2), we get

$$\begin{aligned} 0 = \nabla \cdot \boldsymbol{u}^{n+1} &= \nabla \cdot \frac{\partial}{\partial t}\boldsymbol{U}^n - \Delta t \Delta p^{n+1} \\ \implies \Delta p^{n+1} &= \frac{1}{\Delta t} \nabla \cdot \frac{\partial}{\partial t}\boldsymbol{U}^n. \end{aligned} \tag{4.10}$$

Therefore p^{n+1} can be obtained through the solution to the Poisson equation (4.10), described in the next section. Note that this implies that the pressure field is chosen precisely so that the continuity equation is satisfied, ensuring that the fluid is incompressible at each time level.

This explicit-implicit formulation motivates the three-step solution procedure for obtaining $\boldsymbol{u}^{\boldsymbol{n+1}}$ at each time step:

1. Compute the provisional velocity field $\boldsymbol{U}^n = \boldsymbol{u}^n + \frac{\partial}{\partial t}\boldsymbol{U}^n \Delta t$ by applying an explicit time integrator to compute the fluid velocities in absence of pressure.

2. Solve the Poisson equation (4.10) to obtain the pressure correction p^{n+1}.

3. Correct the provisional velocity field with pressure gradient ∇p^{n+1} to obtain the incompressible velocity field $\boldsymbol{u}^{n+1}$ using equation (4.6).

The motivation for this splitting technique is that incompressibility is assured at each time step by the pressure correction, and the use of explicit methods for the fluid velocities increases performance for flows with larger Reynolds numbers. This time step is governed by the standard Courant-Friedrichs-Lewy (CFL) condition, a fundamental concept in the explicit numerical solution to PDEs [29]. In this method, the time step Δt is constrained by a set of conditions, coming from the diffusive, and convective terms respectively [18],

$$2\frac{\Delta t}{Re} < \Big(\sum_{i=1,2,3} \frac{1}{\Delta x_i^2}\Big)^{-1}, \quad |\max\{\boldsymbol{u}_i\}|\Delta t < \Delta x_i, i = 1, 2, 3. \tag{4.11}$$

Additionally, we avoid the use of a Newton-type iterative procedure necessary for fully implicit methods, which may come at a substantial performance cost. Instead, we perform a single linear solve on each time step to update the solution of the Poisson equation (4.10), which can be done efficiently using standard iterative techniques. The primary downside to this approach in contrast to fully implicit methods is that to maintain numerical stability for high-Reynolds number flows we must impose fairly strict stability conditions on the time step, as can be seen in the first condition in (4.11).

4.1.2 Spatial Discretization

To perform accurate and efficient spatial discretizations for the Navier-Stokes equations, we use the staggered grid arrangement for the positions of the unknowns on the spatial domain Ω_f^d (reference here), a standard approach in fluid simulation. The staggered grid is a typical uniform grid partitioning the domain in each axis, however, each of the dependent variables in each cell are placed in different locations within each grid cell. This can be seen for the two and three-dimensional cases in Figures 4.1 and 4.2, respectively. In the two-dimensional case, the u and v components of the fluid velocity in this cell are placed in the center of the northern and eastern edges of the cell respectively, and the fluid pressure is placed at the cell center, and the three-dimensional staggered grid cell is defined analogously. The advantage of this grid arrangement is that it allows us to use second-order central difference schemes to discretize the Navier-Stokes equations while avoiding phenomena such as the checkerboard instability [7].

In this section, we describe the spatial discretization of the momentum equation (4.1) on a staggered grid. We proceed primarily from standard methods detailed in [18] and

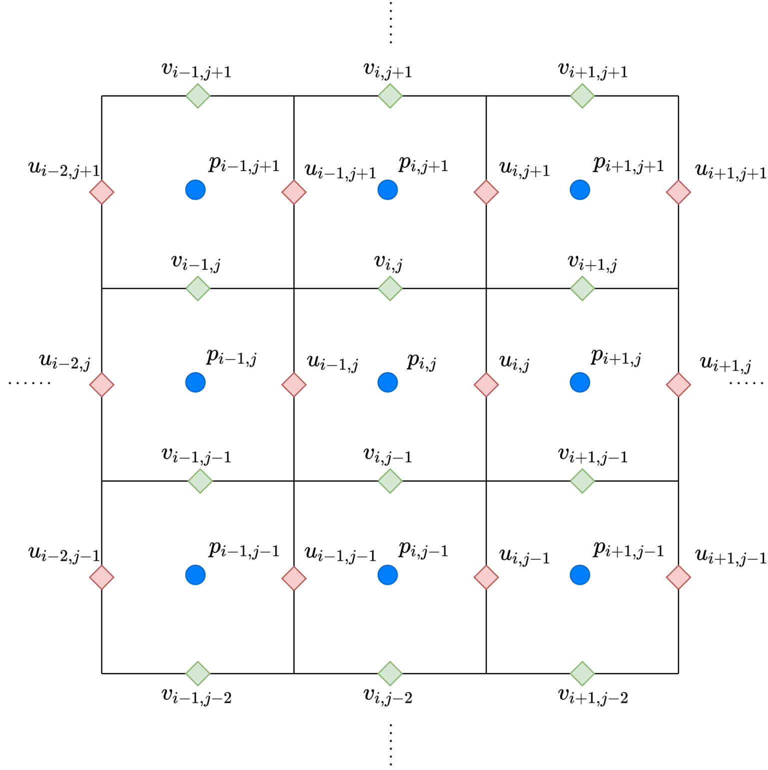

Figure 4.1: 2D Staggered Grid

[7]. The terms of the Navier-Stokes equations are divided into three classes relevant to the numerical solution to this equation, the diffusive terms $\Delta \boldsymbol{u}$, the advective terms $(\boldsymbol{u} \cdot \nabla)\boldsymbol{u}$ and the pressure terms ∇p. We explain the discretizations for each velocity term in the two-dimensional case, noting that the generalization to three dimensions is quite natural. The components of the two-dimensional velocity vector as $\boldsymbol{u} = \begin{bmatrix} u & v \end{bmatrix}^T$. The two-dimensional stencil upon which each discretization is applied in each cell is shown in Figure 4.3. Applying the forward Euler method to integrate the equation (4.6), we get the update rules

$$u^{n+1} = F^n - \Delta t \frac{\partial p^n}{\partial x},$$
$$v^{n+1} = G^n - \Delta t \frac{\partial p^n}{\partial y}, \tag{4.12}$$

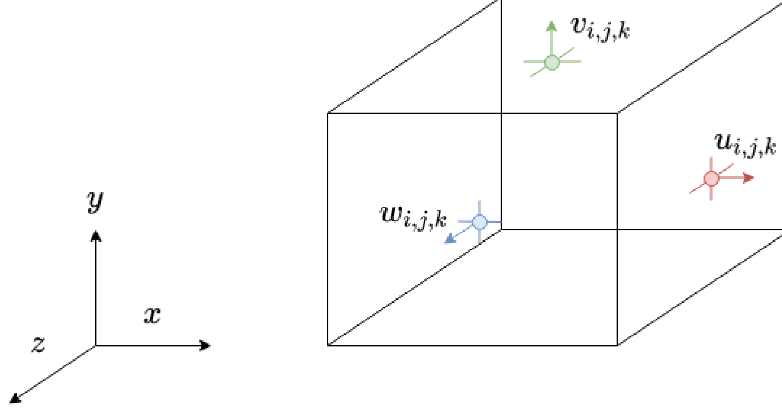

Figure 4.2: 3D Staggered Grid Cell

where

$$F^n = u^n + \Delta t\Big[\frac{1}{Re}\Big(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\Big) - \frac{\partial u^2}{\partial x} - \frac{\partial (uv)}{\partial y} + g_x\Big],$$
$$G^n = v^n + \Delta t\Big[\frac{1}{Re}\Big(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\Big) - \frac{\partial v^2}{\partial y} - \frac{\partial (uv)}{\partial x} + g_y\Big].$$

For the discretization of the diffusive terms $\nabla \boldsymbol{u} = \frac{\partial^2 \boldsymbol{u}}{\partial x^2} + \frac{\partial^2 \boldsymbol{u}}{\partial y^2}$, we use the centered finite difference schemes

$$\frac{\partial^2 u^n_{i,j}}{\partial x^2} \approx \frac{u^n_{i+1,j} - u^n_{i-1,j}}{\Delta x^2}, \tag{4.13}$$
$$\frac{\partial^2 u^n_{i,j}}{\partial y^2} \approx \frac{u^n_{i,j+1} - u^n_{i,j-1}}{\Delta y^2}, \tag{4.14}$$
$$\frac{\partial^2 v^n_{i,j}}{\partial x^2} \approx \frac{v^n_{i+1,j} - v^n_{i-1,j}}{\Delta x^2}, \tag{4.15}$$
$$\frac{\partial^2 v^n_{i,j}}{\partial y^2} \approx \frac{v^n_{i,j+1} - v^n_{i,j-1}}{\Delta y^2}. \tag{4.16}$$

While the advantages of using the staggered grid are clear in terms of stability and accuracy, it does introduce several additional complexities in other parts of the scheme. This can be seen in the convective terms, where the misalignment of the velocity components necessitates the use of interpolation to re-align the solution information onto the grid

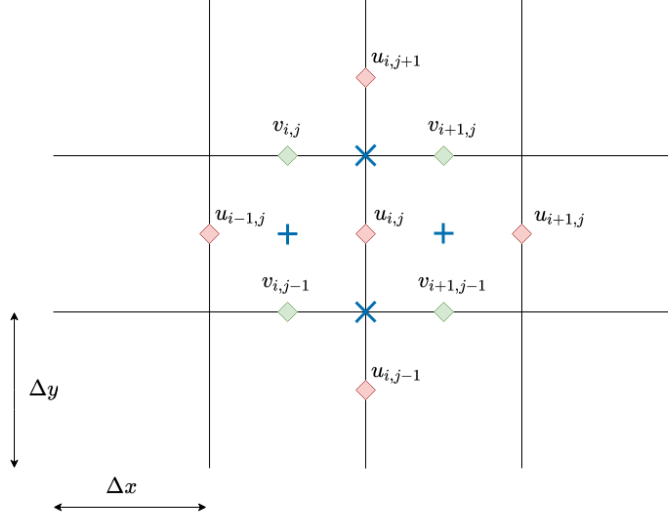

Figure 4.3: 2D Stencil for the velocity terms in the momentum equation, centered at the x-component of the fluid velocity

points. In Figure 4.3, the requisite interpolated values of u and v required to compute the convective terms on the staggered grid are shown, with the points required for the x and y derivatives marked with $+$ and $\times$ respectively. The resulting centred difference scheme is given below

$$\frac{\partial (u^n_{i,j})^2}{\partial x} \approx \frac{(u^n_{i+\frac{1}{2},j})^2 - (u^n_{i-\frac{1}{2},j})^2}{\Delta x}, \tag{4.17}$$

$$\frac{\partial (v^n_{i,j})^2}{\partial y} \approx \frac{(v^2_{i,j+\frac{1}{2}})^n - (v^2_{i,j-\frac{1}{2}})^n}{\Delta y}, \tag{4.18}$$

$$\frac{\partial (uv^n_{i,j})}{\partial x} \approx \frac{u^n_{i,j+\frac{1}{2}} v^n_{i+\frac{1}{2},j} - u^n_{i-1,j+\frac{1}{2}} v^n_{i-\frac{1}{2},j}}{\Delta x}, \tag{4.19}$$

$$\frac{\partial (uv^n_{i,j})}{\partial y} \approx \frac{u^n_{i,j+\frac{1}{2}} v^n_{i+\frac{1}{2},j} - u^n_{i,j-\frac{1}{2}} v^n_{i+\frac{1}{2},j-1}}{\Delta y}, \tag{4.20}$$

where we denote $f_{i+\frac{1}{2}} = \frac{f_i + f_{i+1}}{2}$. Note that the location of these derivatives are chosen to align with the location of the respective velocity components in each fluid cell.

In order to stabilize the solution of the convective terms, [18] suggests the use of a cell-donor scheme, wherein we combine these centred schemes with upwind schemes to improve the stability of the discretization. Bridson [7] recommends the use of a semi-Lagrangian

scheme, wherein a Runge-Kutta integrator is used to compute the previous location of imaginary particles passing through each grid cell for use in the discretization. Other methods include those based on Essentially Non-Oscillatory schemes [36], which are high order schemes which maintain stability by adaptively choosing from a set of stencils the one which will have the smallest in magnitude derivatives, guaranteeing the most stable finite difference scheme possible for a given order of accuracy. We make use of the simple cell donor scheme, which reduces the order of accuracy of the discretization of the convective terms to first order but is simple and effective, particularly in the case of a constantly changing fluid domain, where stencil points may vanish at any time, necessitating the use of variable order approximations at each grid cell if higher order is desired.

For the pressure derivatives in (4.12), we apply the simple centered finite difference schemes

$$\frac{\partial p^n_{i,j}}{\partial x} = \frac{p^n_{i+1,j} - p^n_{i,j}}{\Delta x}, \tag{4.21}$$

$$\frac{\partial p^n_{i,j}}{\partial y} = \frac{p^n_{i,j+1} - p^n_{i,j}}{\Delta y}. \tag{4.22}$$

To obtain the pressure values within each grid cell, we must solve the Poisson equation (4.10), which is given in the 2D case by

$$\frac{\partial^2 p^n}{\partial x^2} + \frac{\partial^2 p^n}{\partial y^2} = \frac{1}{\Delta t}\Big(\frac{\partial F^n}{\partial x} + \frac{\partial G^n}{\partial y}\Big). \tag{4.23}$$

The Poisson equation is discretized using the centered finite difference scheme

$$\frac{\partial^2 p^n}{\partial x^2} = \frac{p^n_{i+1,j} - 2p^n_{i,j} + p^n_{i-1,j}}{\Delta x^2}, \tag{4.24}$$

$$\frac{\partial^2 p^n}{\partial y^2} = \frac{p^n_{i,j+1} - 2p^n_{i,j} + p^n_{i,j-1}}{\Delta y^2}. \tag{4.25}$$

This discretization results in a symmetric positive definite system of linear equations, which is solved using the conjugate gradient method.

4.1.3 Boundary Conditions

In the previous section, we discussed the discretization of the momentum equation. To completely specify the solution, we must apply boundary conditions. The use of a staggered

grid complicates the application of boundary conditions, as there may be no solution information on the boundary itself. To resolve this issue, we introduce the concept of a ghost cell, additional cells which are added so that many kinds of boundary conditions can be more easily applied on the staggered grid. The ghost cells on the corner of a two-dimensional staggered grid are visualized in Figure 4.4. These ghost cells are used in two ways: setting boundary conditions when the unknown velocity components are aligned with the boundary, and interpolating the boundary conditions when they do not align. For simplicity, in this chapter, we first consider setting the boundary conditions on a two-dimensional domain with a simple rectangular boundary.

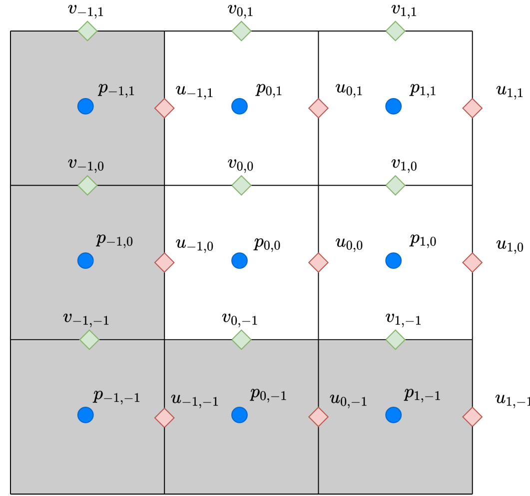

Figure 4.4: 2D Ghost Cells

For the velocity components, there are a few common boundary conditions that are used in the context of fluid modelling. The *no-slip* condition states that the fluid velocities must vanish at the boundaries, that is, $\boldsymbol{u}(\boldsymbol{x}) = \mathbf{0}$ for all $\boldsymbol{x} \in \Gamma_t$. We use no-slip for the outer boundaries of our simulations, e.g., on the non-object boundaries of the domain. To demonstrate how to implement this condition on the staggered grid, we consider the case where the velocity component lays on the boundary and the case where it does not. In the first case, we can simply set the value in the ghost cells on the rectangular boundary as

$$\begin{aligned} u_{-1,j} = 0, \quad u_{n_x-1,j} = 0, \quad j = 0, \ldots, n_x - 1, \\ v_{i,-1} = 0, \quad u_{i,ny-1} = 0, \quad i = 0, \ldots, n_y - 1. \end{aligned}$$

When the components in the ghost cell are not aligned with the boundaries, we use interpolation to enforce the boundary conditions. This is done by assigning an appropriate value in the ghost cell. For example, to assign the no-slip condition in the cell $(0, 0)$ to the value $u_{0,-1}$, we interpolate using the nearest velocity value normal to the boundary, $u_{0,0}$ using the formula

$$u_\Gamma = \frac{u_{ghost} + u_{normal}}{2} \tag{4.26}$$

$$= \frac{u_{0,-1} + u_{0,0}}{2}. \tag{4.27}$$

Then assigning the no-slip condition on the boundary, $u_\Gamma = 0$, we use the above formula to find that $u_{ghost} = -u_{normal}$. On the rectangular domain, this leads to the set of boundary conditions

$$v_{-1,j} = -v_{0.j}, \quad v_{n_x,j} = -v_{nx,j}, \quad j = 0, \dots, n_x - 1$$
$$u_{i,-1} = -ui, 1, \quad u_{i,n_y-1} = -u_{i,n_y}, i = 0, \dots, n_y - 1.$$

This is the standard approach to set boundary conditions on the staggered grid, and the other types of boundary conditions make use of similar methods.

To model phenomena such as liquid flowing through a pipe, we define *inflow* and *outflow* conditions. For inflow conditions, we explicitly state the velocity of the fluid entering the region through the boundaries. We do this using the same procedure as we did for the no-slip condition, setting the values explicitly when the velocities align with the boundaries and using interpolation with the ghost cells otherwise. With outflow conditions, we specify that the normal derivatives of both velocity components must be 0, that is $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}} = \mathbf{0}$.This condition essentially specifies that the fluid at this boundary point has left the domain, and should no longer change the flow. These are used in several of our example simulations.

There are several other common boundary conditions that we do not consider in this thesis. Free-slip conditions specify that only the velocity component normal to the boundary should vanish, and the velocity component tangential to the boundary should be unchanged. Periodic boundary conditions are also common [7].

We must also specify boundary conditions for the Poisson equation (4.10). To handle this, we make use of the Chorin projection method [9], where we multiply the discretized momentum equation (4.6) by the outward normal vector $\boldsymbol{n}$ to get

$$\nabla p^n \cdot \boldsymbol{n} = -\frac{1}{\Delta t}\Big(\boldsymbol{u}^n - \boldsymbol{F}^n\Big) \cdot \boldsymbol{n}. \tag{4.28}$$

This is a Neumann type boundary condition, meaning that the discretization of the Poisson equation (4.10) results in a singular linear system. However, it is known that when solved using the conjugate gradient method, it is guaranteed to converge [51].

4.2 Interface Representation

For several components of the algorithm, it is necessary that we maintain a distinction between parts of the simulation domain occupied by fluid or solid material. One of the most popular approaches for accomplishing this is the use of implicit representations of the objects' boundaries. In an implicit representation, we maintain a level set function ϕ to keep track of the location and shape of complex interfaces. The textbooks [36], [16] and [44] provide excellent introductions to level set methods and associated numerical algorithms. The spatial domain for the simulation is $\Omega^d \subset \mathbb{R}^d$. Defining Ω_t^f as the subset of Ω^d containing fluid at time t, and Ω_t^s as the portion of the domain containing immersed structures, and $\Omega_t^f \cap \Omega_t^s = \Gamma_t$ being the fluid-solid interface at the current point in time, the level set function is defined as

$$\phi(x) : \begin{cases} > 0 & x \in \Omega_t^f \\ < 0 & x \in \Omega_t^s \\ = 0 & x \in \Gamma_t \end{cases} . \tag{4.29}$$

In the discrete setting, where the level set function is defined on a numerical grid, a grid cell is in the fluid region if the value of $\phi(x)$ at its center is positive, in an object region if the value is negative, and is on the object-fluid interface if the sign changes from positive to negative within the interior of this cell. See Figure 4.6 for an example of how a level set function can be used to represent a circle in two dimensions, and Figure 4.5 for how this is labelled on our Euclidean grid. This approach was popularized in [37] and has become a standard approach in fluid dynamics, as well as in many other application areas such as computer graphics and computer vision [45, 48, 7]. The primary utility of this approach is that complex shapes can be represented on a uniform mesh by using interpolation to recover the details at each point. This has a substantial advantage in terms of complexity over algorithms which must manage complex unstructured meshes to maintain boundary information. Additionally, there are many useful operations that we can apply to the level set function to very efficiently manipulate it.

While the zero-isocontour of a level set function can be used to represent a particular shape, in general, there are infinitely many such functions having the same zero-isocontour.

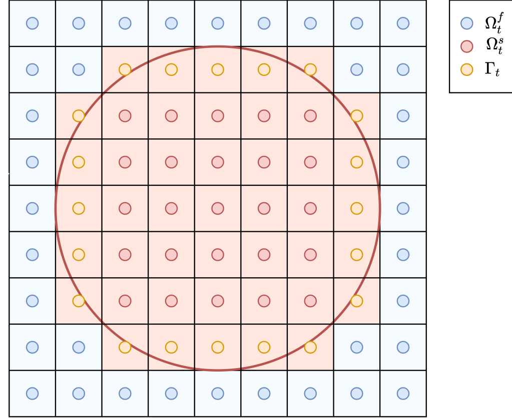

Figure 4.5: Grid labelling of grid cells as solid and fluid based on value of signed distance function for a circle.

Therefore, an important question becomes which is the *ideal* function to approximate the shape in question. Due to its many useful properties, the *signed distance function* of a particular object boundary is typically desired. The signed distance function is a level set function with the added condition that

$$\phi(\boldsymbol{x}) = S(\boldsymbol{x}) \min_{\boldsymbol{x}_I \in \Gamma}\{\boldsymbol{x} - \boldsymbol{x}_I\}, \quad S(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{x} \in \Omega_t^f \\ -1 & \boldsymbol{x} \in \Omega_t^s \end{cases}. \tag{4.30}$$

That is, for each $\boldsymbol{x} \in \Omega^d$, $\phi(\boldsymbol{x})$ is simply the distance of x from the nearest object interface. To guarantee that a level set function ϕ is a signed distance function, we ensure that the Eikonal equation, referred to as the reinitialisation equation,

$$||\nabla\phi(\boldsymbol{x})||_2 = 1, \tag{4.31}$$

is satisfied for all $\boldsymbol{x} \in \Omega^d$. There are several standard methods for solving this equation, several of which are described in the texts [36] and [44]. Notable additions to this work include [8] and [42] which discuss algorithmic refinements which improve stability and accuracy when computing signed distance functions.

When using the level set representation of a surface, we must be able to reconstruct features of the object boundary by using the local information which is available on each

grid cell. An important property of the signed distance function is that, given a point $\boldsymbol{x}$, it is simple to find the nearest point on the interface to $\boldsymbol{x}$. This is done using the formula

$$\boldsymbol{x}_c = \boldsymbol{x} - \phi(\boldsymbol{x})\frac{\nabla\phi(\boldsymbol{x})}{||\nabla\phi(\boldsymbol{x})||_2}. \tag{4.32}$$

where $\boldsymbol{x}_c$ is the unknown nearest point to $\boldsymbol{x}$. To demonstrate the difference between a more standard level set representation, and the signed distance representation, we plot the difference between a two-dimensional circle represented as both the zero-isocontour of a level set function using the equation of the circle and as the zero-isocontour using the equation of a cone, which is the signed distance representation in Figure 4.6.

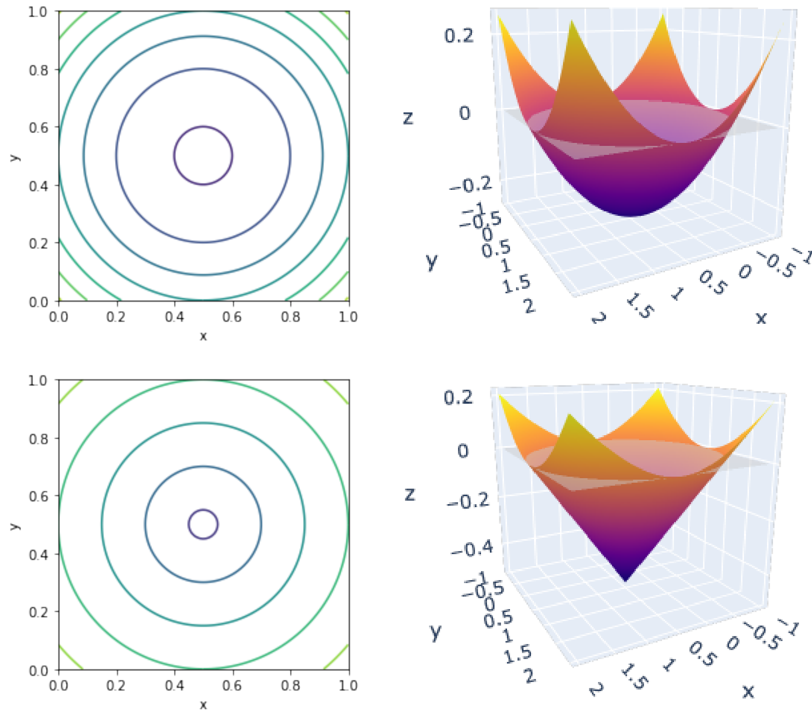

Figure 4.6: Two different representations of a circle using the level set method. First row is a circle attained using the level set function corresponding to the standard equation of a circle $\phi(x, y) = x^2 + y^2 - 0.5^2$. Second row is attained using the level set function corresponding to the equation of a cone $\phi(x, y) = \sqrt{x^2 + y^2} - 0.5$, the signed distance representation of a circle.

4.2.1 Interface Motion

In our simulation, the interfaces of each of the solid objects are constantly in motion and undergoing deformation. Fortunately, the implicit representation of the interfaces allows us to, relatively simply, move the distance field, rather than recreating it completely at each time step, which is in general prohibitively expensive. Let $V : \Omega^d \mapsto \mathbb{R}^d$ be the velocity field that specifies the deformation of the level set function at each point in the domain. We wish to move the level set such that for each material point on the domain we have $\frac{d\boldsymbol{x}}{dt} = \boldsymbol{V}$ and the value $\phi(\boldsymbol{x}) = 0$ is maintained. Applying this logic across not only the zero-isocontour, we state that each point on the domain should move with velocity $\boldsymbol{V}$ and keep its value of ϕ constant, giving the equation $\frac{D\phi}{Dt} = 0$, which can then be re-written as

$$\phi_t(\boldsymbol{x}, t) + \boldsymbol{V} \cdot \nabla\phi = 0. \tag{4.33}$$

This equation is related to the standard transport equation [29], and therefore similar numerical methods are used to solve it. In particular, the use of methods for Hamilton-Jacobi equations is discussed in detail in [36] for solving this equation stably with high orders of accuracy. We make use of a third-order ENO scheme to discretize the gradient in (4.33), and the third-order *Total Variation Diminishing* (TVD) Runge-Kutta scheme for stable explicit time stepping. TVD schemes ensure the least oscillatory time stepping of a particular order, assuming certain standard restrictions on step size are maintained [17] (with the first order TVD Runge-Kutta method, the standard forward Euler's method, being monotone and hence completely non-oscillatory).

The use of a third order ENO scheme for spatial discretization in conjunction with the TVD Runge-Kutta time integration ensures that this method is the least oscillatory third order finite difference scheme possible, and hence the most stable.

Note that in this formulation we move the level set function at each point in the domain Ω^d, however, the only points of interest are actually the zero-isocountours. The reason for this goes back to the theory of hyperbolic PDE's [29] where we can deduce from the method of characteristics that when neighbouring velocities differ, it can result in discontinuities in the solution which can then damage the zero-isocontour. Therefore, at least a small band around the zero-isocontour must be moved with a continuous velocity field to avoid the effects of these discontinuities.

4.2.2 The Fast Marching Method

For the operation of reinitializing a level set function to a signed distance function we must solve the equation (4.31). Additionally, for reasons that will be clear in the next

chapter, given a set of velocities field $\boldsymbol{V}_{\phi(\boldsymbol{x})=0}$ defined on the zero-isocontour of the level set function, we would like to extrapolate this velocity field normal to the zero-isocontour into the remainder of the domain. This is so that it can be used within the transport equation (4.33) to move the level set function.

The most basic methods use time relaxed versions of the reinitialization in the form of Hamilton-Jacobi type PDEs to gradually converge towards a solution [36]. These approaches allow us to easily make use of standard methods to generate approximate solutions but solving these equations near-enough to steady-state may be prohibitively expensive due to stability restrictions on the time step for explicit methods. Sethian [43] introduced the concept of the Fast Marching Method (FMM), a method for directly solving general Eikonal equations of the form

$$||\nabla u||_2 = F. \tag{4.34}$$

For example, with $F = 1$, and $u = \phi$ we get the reinitialization equation (4.31). For extrapolating the fluid velocity, the method presented in [1] adds an extra step to the FMM procedure, gradually building up the best-possible normally extrapolated velocity field over the course of the algorithm. We present the combined algorithm below, with some intuition from the theory of characteristics that motivates the algorithm.

The algorithm begins by marking which grid cells contain interfaces, that is, which cells contain both fluid and solid material. Figure 4.5 can be used as a reference. In each interface cell, we store each cell in order by minimum ϕ value, which can be obtained in several ways, primarily by interpolating the boundary of the previous level set function, though we just approximate the actual distance of the cell center to the object surface. The grids are stored using a minimum heap for fast addition and removal of new cells. The interface nodes are labelled CLOSE, aside from the node with the least signed distance, which is labelled TRIAL. All of the remaining nodes are labelled FAR, and once a node has been processed by the algorithm, we label it ACCEPTED.

The main idea of the FMM is to use available knowledge about the distance to the interface to construct the next level of information, by maintaining and adjusting the estimates for the signed distance function. On each step, we pop the grid point with the least signed distance from the heap and label it as the new TRIAL point. We then update the estimate of the signed distance in each neighbouring grid cell in class FAR [7].To update the estimates of the signed distance, we make use of the discretization of the Eikonal equation (4.31), where the discretization stencil is dependent on which solution information has already been computed before the current step. In two dimensions, the

discretization for the signed distance equation is given by

$$(D_x^i \phi)^2 + (D_y^i \phi)^2 = 1, \tag{4.35}$$

where D_w^i is the upwind differentiation operator, given by

$$D_w^i = \begin{cases} \frac{\phi_{i+1}-\phi_i}{\Delta w} & \phi_{i+1} \leq \phi_{i-1}, \\ \frac{\phi_i - \phi_{i-1}}{\Delta w} & \text{otherwise.} \end{cases} \tag{4.36}$$

When the proper upwind direction is not available (the corresponding cell is not in ACCEPTED), we simply use the other direction, and when neither upwind direction is available for a particular axis, we assume that the derivative along this axis is 0. To determine the values of ϕ on each step, we follow the algorithm presented in Bridson [7], which uses the presently available information at the present step, for the current grid cell in order to produce the best available approximation at this step.

Sethian and Adelstein [1] add an additional step to the FMM algorithm in order to perform normal extrapolation and build the signed distance function simultaneously. To do this, we solve the discretized form of the equation

$$\nabla \phi \cdot \nabla F = 0. \tag{4.37}$$

In two dimensions, this discretization is given by

$$\begin{bmatrix} D_x^i \phi & D_y^i \phi \end{bmatrix} \begin{bmatrix} D_x^i F & D_y^i F \end{bmatrix}^T = 0, \tag{4.38}$$

where the upwind operators are defined as in (4.36). This leads to the update rule

$$F_{i,j} = \frac{F_{i+u_x,j} \Delta x D_x^i \phi + F_{i,j+u_y} \Delta y D_y^j \phi}{\Delta x D_x^i \phi + \Delta y D_y^j \phi} \tag{4.39}$$

The three dimensional generalization can be derived in an identical way.

```
Result: A reinitialized signed distance function φ, and normally extrapolated
        velocity field V
Data: All cells have status FAR, all φ and V velocities set to infinity.
// Label all of the cells and add interface cells to the heap
for cell in Ω_t^f do
    if cell.type == INTERFACE then
        heap.push(cell);
        cell.status = CLOSE;
        cell.phi = approximateDistOnInterface(cell);
        cell.V = approximateVelOnInterface(cell);
    end
end
// Run the fast marching algorithm in the fluid domain
while !heap.empty() do
    trial_cell = heap.pop();
    trial_cell.status = TRIAL;
    // Update approximate distance field and velocities in
       neighbouring cells.
    for cell in trial_cell.neighbours do
        if cell.type == FAR then
            cell.phi = approximateDist(cell);
            cell.V = approximateVel(cell);
            cell.status = FAR;
            heap.push(cell);
        end
    end
end
```

Algorithm 1: Signed distance reinitialization with velocity extrapolation using the Fast Marching Method. This algorithm is applied twice, once in Ω_t^f, once in Ω_t^s.

We make use of only a single signed distance function to represent the fluid-solid interface of all objects in the domain. This is primarily for efficiency purposes, to avoid the necessity of having to build and integrate new interface locations for each object on each time step. When the number of objects is large, this may become prohibitively expensive. This does, however, introduce a large number of difficulties into the system. To start with, if two interfaces happen to intersect each other over the course of the simulation, this will

always lead to a loss of boundary information for each of these two objects. Therefore, our algorithms from this point are designed such that the various zero-isocontours of the objects should not come into contact during the course of the simulation.

4.3 Deformable Solid Simulation

For large-scale FSI problems with many object interactions, it is well documented that the cost of handling the structural component of the simulation can come to dominate the overall cost [24]. Commonly in engineering type simulations, a Finite Element Method (FEM) is used to solve the PDEs which model constitutive properties of the material based on the internal and external stress acting on the structure at each point in time, the elements forming the simulated structure are deformed based on the estimated strain. While FEM can produce state-of-the-art results in terms of accuracy, it can be very costly, particularly when many objects must be simulated. As an alternative, we make use of spring assemblies as the basis for our FSI algorithm. In this section, we describe the theory and some standard approaches for solid modelling in CFD before describing our algorithm for quick simulation of many interacting deformable solids. The physical theory for the deformable solids was previously introduced in 3.2.

4.3.1 Approximation Using Mass-Spring Assemblies

As mentioned previously, while FEM methods can accurately model deformable objects, they can be very expensive in general, prohibitively so in the context of many object FSI simulations. To address this, we make use of an approximation of the continuous models for material deformation by instead modelling our solids as assemblies of mass particles connected by linear elements, referred to commonly as Mass-Spring Systems (MSS's). This is a common simplification applied in engineering and real-time applications, as rather than solving complicated non-linear PDEs using FEM to obtain the deformations, an approximating system of ODEs can instead be solved using well-tuned numerical methods. In particular, this model has seen much use in computer graphics, where it can be employed to model real-time deformations, or to impose geometric constraints on evolving scene objects [6]. Another avenue for which MSS's have been used to great success is in cloth simulation, where the use of interacting point masses dramatically decreases the complexity in modelling complicated contact problems, such as draping a simulated cloth over a character model [31]. For real-time surgery simulations, [50] uses viscoelastic MSS's in the context of real-time surgery simulation. In the context of FSI, [15] gives a grid-based mass-spring

model based on the MAC method, and the blood flow simulation [52] uses a mass-spring model to integrate many interacting blood cells being transported in a capillary flow in a particle-based simulation of blood flow. Two and three dimensional MSS's are visualized in Figure 4.7

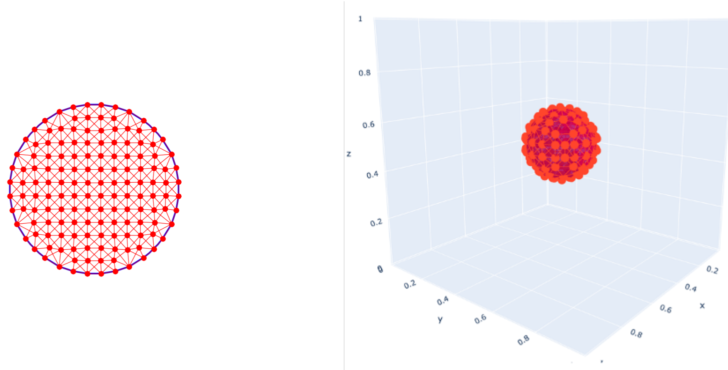

Figure 4.7: Mass-Spring System assembly for a circle and a sphere.

In our simulation, we make use of two kinds of spring elements, linear and torsion springs. These correspond respectively with the linear and deviatoric components of the Cauchy strain tensor (3.20) which is illustrated in Figures 4.8 and 4.9 for the two and three-dimensional cases, respectively. For the linear element connecting the ith and jth point masses, we make use of simple constitutive equations of the form

$$\sigma_{ij} = F(\epsilon_{ij}), \tag{4.40}$$

where $\epsilon_{ij} = ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2 - l_{ij}$ is the strain of this element and l_{ij} is the resting length of the spring. The most simple of these constitutive equations, and the one we apply in our simulation is the elastic spring element, which is governed by Hooke's law

$$\sigma_{ij} = E\epsilon_{ij}, \tag{4.41}$$

where E is Young's modulus [14]. To add additional dynamics to the structural model, such as viscoelasticity, one can use alternative constitutive models, though these typically add additional non-linearity which must be addressed when solving these systems. One of these alternatives is the linear dashpot model, which models viscosity, and has the constitutive equation

$$\sigma_{ij} = \eta \frac{d\epsilon_{ij}}{dt}, \tag{4.42}$$

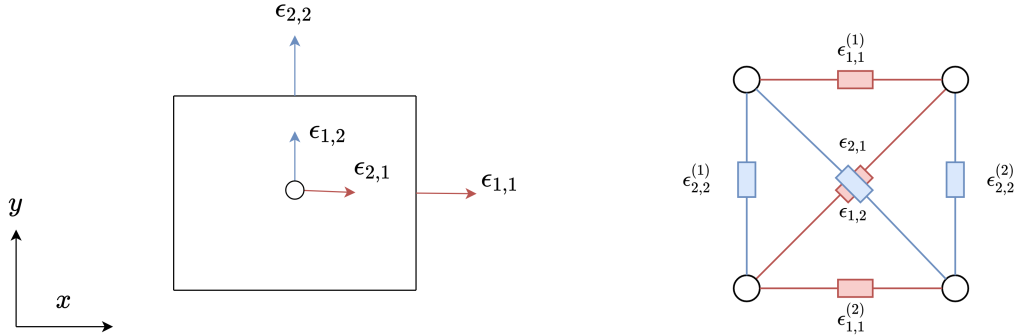

Figure 4.8: (Left) The components of the two-dimensional Cauchy strain tensor in terms of how they affect the deformation of a solid volume. (Right) The analogous two-dimensional MSS, where superscripts indicate some that two edges may have duplicate roles in terms of the analogy between the two..

where η is the coefficient of viscosity. The linear spring and linear dashpot elements are visualized in Figure 4.10.

Additional constitutive equations can be generated as sequences and series of these simple linear spring and dashpot elements. The Maxwell model is formed by joining a linear spring and dashpot in series. Due to this configuration, the strain for each element is the same, but the stresses will be different. The resulting strain in the element is simply the sum of each element,

$$\epsilon_{\text{maxwell}} = \epsilon_{\text{spring}} + \epsilon_{\text{dashpot}}. \tag{4.43}$$

Differentiating (4.41) and (4.43) with respect to time, we get the equations

$$\frac{d}{dt}\sigma = E\frac{d}{dt}\epsilon_{\text{spring}}, \tag{4.44}$$

$$\frac{d}{dt}\epsilon_{\text{maxwell}} = \frac{d}{dt}\epsilon_{\text{spring}} + \frac{d}{dt}\epsilon_{\text{dashpot}}. \tag{4.45}$$

Combining (4.42) and (4.44) into (4.45), we obtain the following differential equation which represents the constitutive behaviour of a linear viscoelastic solid,

$$E\frac{d\epsilon}{dt} = \frac{d\sigma}{dt} + \frac{E}{\eta}\sigma, \tag{4.46}$$

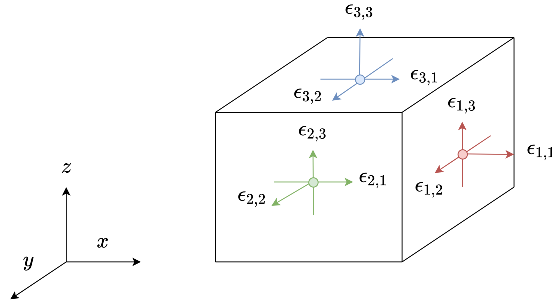

Figure 4.9: Visualization of the effect of the three-dimensional Cauchy Stress tensor on a volume.

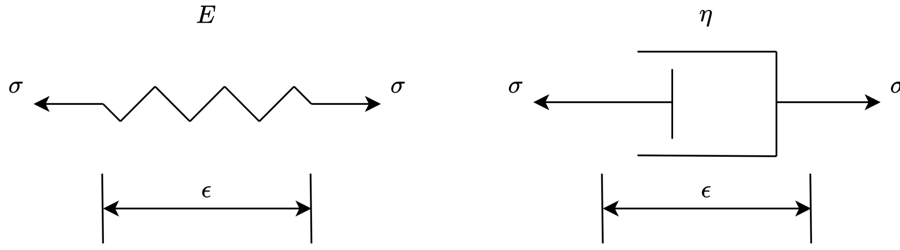

Figure 4.10: Spring and Dashpot Linear Element Diagrams.

The Kelvin model is another standard alternative for modelling viscoelasticity, where rather than connecting the spring and dashpot in series, we connect them in parallel. As a result, the stresses for each element will be different, but the strains are identical. Therefore the total stress will be additive, written as

$$\sigma_{\text{kelvin}} = \sigma_{\text{spring}} + \sigma_{\text{dashpot}}, \tag{4.47}$$

and making similar substitutions as in the Maxwell model, we obtain the constitutive differential equation

$$\eta \frac{d\epsilon}{dt} + E\epsilon = \sigma. \tag{4.48}$$

The Maxwell and Kelvin models are visualized in Figure 4.11. Further generalization of these models typically involve adding additional Maxwell and Kelvin models in sequence and series. For more information, see [11].

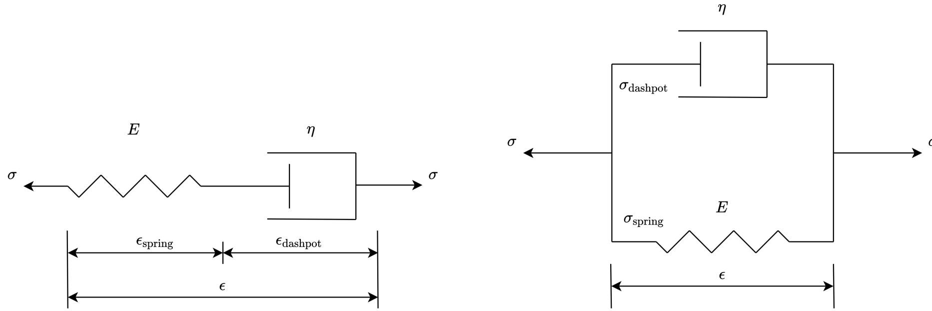

Figure 4.11: Maxwell and Kelvin Model Diagrams.

4.3.2 Modelling Mass-Spring Systems

We now discuss how to model the deformation of the Mass-Spring Systems introduced in the previous section. Note that our formulation is based on that provided from the excellent course notes [30]. We first introduce some notation. Let N be the number of nodes in the Mass-Spring Assembly. Then we assemble all of the mass point coordinates into the vector $\boldsymbol{q} \in \mathbb{R}^{d\times N}$. Let M be the bulked mass matrix $M = mI$, that is, we suppose that the mass is the same for each point in the MSS, $m = m_{\text{obj}}/N$. and I is the identity matrix. We first consider how to integrate a single spring, before moving onto the case of the spring assembly.

The potential energy for a single Hookean spring follows the constitutive equation (4.41)

$$V = \frac{1}{2}E(l - l_0)^2, \tag{4.49}$$

where l is the current length of the spring and l_0 is its resting length. To express this in terms of the vector q (in this case $\boldsymbol{q} = \begin{bmatrix}\boldsymbol{x}_0^T & \boldsymbol{x}_1^T\end{bmatrix}^T$), we introduce the selection matrix D, defined as

$$D = \begin{bmatrix}-I & I\end{bmatrix}, \tag{4.50}$$

which allows us to write the length of the spring

$$l = ||\boldsymbol{x}_1 - \boldsymbol{x}_0||_2 = \sqrt{\boldsymbol{q}^T D D^T \boldsymbol{q}}, \tag{4.51}$$

and hence $V(\boldsymbol{q}) = \frac{1}{2}E(\sqrt{\boldsymbol{q}^T D^T D\boldsymbol{q}} - l_0)^2$. To move the particles, we apply Newton's second law, for which we need to determine the acceleration of each node within the system. To do

this, we consider the variational perspective for the mechanics of the MSS. The approach that we use is called the Principle of Least Action, which asserts that the correct trajectory of an object over time is the one that minimized the difference between the potential and kinetic energy within the system,

$$\boldsymbol{q}(t) = \text{argmin}_{\boldsymbol{q}} \int_{t_0}^{t_1} T(\boldsymbol{q}, \dot{\boldsymbol{q}}) - V(\boldsymbol{q}, \dot{\boldsymbol{q}}) dt, \tag{4.52}$$

where T is the kinetic energy and V is the potential energy, with the quantity $L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T(\boldsymbol{q}, \dot{\boldsymbol{q}}) - V(\boldsymbol{q}, \dot{\boldsymbol{q}})$ being known as the Lagrangian of this system. The solution of this functional for the node locations is then solved using the Calculus of Variations, which states that the correct node locations will be the solution of the Euler-Lagrange equation,

$$\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\Big) = -\frac{\partial L}{\partial \boldsymbol{q}}. \tag{4.53}$$

In the case of a single spring, the kinetic energy for the system is given as $T(\dot{\boldsymbol{q}}) = \frac{1}{2} m \dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}}$, and therefore, into (4.53) for a single spring, the equations of motion are

$$m\ddot{\boldsymbol{q}} = -\frac{\partial V(\boldsymbol{q})}{\partial \boldsymbol{q}}, \tag{4.54}$$

where the derivatives can be easily derived analytically.

We now consider the case of a system of m springs connecting n mass points. These are assembled into a vector $\boldsymbol{q} = \begin{bmatrix} \boldsymbol{x}_0^T & \dots & \boldsymbol{x}_n^T \end{bmatrix}^T$. The kinetic energy is only slightly modified to include the possibility of the mass particles having different masses (though we do not use this generality in this work),

$$T(\dot{\boldsymbol{q}}) = \frac{1}{2} \dot{\boldsymbol{q}}^T M \dot{\boldsymbol{q}}. \tag{4.55}$$

The potential energy is modified to include the potential energy contribution of each of the springs in the system

$$V(\boldsymbol{q}) = \sum_{i=0}^{m-1} V_i(\boldsymbol{x}_0^{(i)}, \boldsymbol{x}_1^{(i)}). \tag{4.56}$$

where $\boldsymbol{x}_0^{(i)}$ and $\boldsymbol{x}_1^{(i)}$ are the generalized coordinate for the points joined by the ith spring. Note the for ith spring, we use the Hookean potential energy (4.49), where the length of

the spring is determined by the distance between the points $\boldsymbol{x}_0^{(i)}$ and $\boldsymbol{x}_1^{(i)}$. For each spring we define the selection matrix D_i as before, however, we modify the dimension to select within the larger matrix, $D_i \in \mathbb{R}^{2d\times n}$, and assemble these into the block diagonal matrix D, where we understand $D\boldsymbol{q}$ to be the a stacked vector of the $\begin{bmatrix}\boldsymbol{x}_1^{(1)} - \boldsymbol{x}_1^{(1)} & \dots & \boldsymbol{x}_N^{(1)} - \boldsymbol{x}_N^{(1)}\end{bmatrix}^T$. Then the expression for the potential energy becomes

$$V(\boldsymbol{q}) = \sum_{i=0}^{m-1} V_i(D_i\boldsymbol{q}). \tag{4.57}$$

Substituting the expression for potential and kinetic energy into the Euler-Lagrange equation, we get the equation of motion

$$M\ddot{\boldsymbol{q}}(t) = \sum_{i=0}^{m-1} D_i^T \frac{\partial V_i(D_i\boldsymbol{q})}{\partial \boldsymbol{x}_0, \boldsymbol{x}_1}. \tag{4.58}$$

We now have an expression which can be used to update the position of the MSS nodes by solving the second order ODE

$$M\ddot{\boldsymbol{q}}(t) = f(\boldsymbol{q}) \tag{4.59}$$

where $f(\boldsymbol{q}) = \sum_{i=0}^{m-1} D_i^T \frac{\partial V_i(D_i\boldsymbol{q})}{\partial \boldsymbol{x}_0,\boldsymbol{x}_1} + \boldsymbol{F}$, where $\boldsymbol{F}$ is the sum of all external forces acting on the solid. More precisely, the total force $\boldsymbol{F} = \boldsymbol{F}_{\text{hydro}} + \boldsymbol{F}_{\text{collision}} + \boldsymbol{F}_{\text{body}}$, where $\boldsymbol{F}_{\text{hydro}}$ is the hydrodynamic stresses due to the action of the fluid on the solid, $\boldsymbol{F}_{\text{collision}}$ is the forces which occur when two different objects are colliding with each other, and $\boldsymbol{F}_{\text{body}}$ are the body forces acting on this object.

4.3.3 Integrating Mass-Spring Systems

To integrate the ODE (4.59) we consider some modern methods that have been applied recently in the field of computer graphics. This method for integrating the MSS constitutive equations was introduced in [38]. We proceed by their derivation of an optimization-based approach for rapidly solving (4.59). We first convert the second order differential equation into the equivalent system of first order equations, which is discretized using the backwards Euler method

$$\boldsymbol{q}^{n+1} = \boldsymbol{q}^n + \Delta t \boldsymbol{v}^n, \tag{4.60}$$
$$\boldsymbol{v}^{n+1} = \boldsymbol{v}^n + \Delta t M^{-1} f(\boldsymbol{q}^n). \tag{4.61}$$

Using the equation (4.60) for two consecutive time steps, we get the expressions for the velocities

$$\Delta t \boldsymbol{v}^n = \boldsymbol{q}^n - \boldsymbol{q}^{n-1}, \tag{4.62}$$

$$\Delta t \boldsymbol{v}^{n+1} = \boldsymbol{q}^{n+1} - \boldsymbol{q}^n, \tag{4.63}$$

which we use to eliminate the velocities in equation (4.61) to obtain

$$\boldsymbol{q}^{n+1} - 2\boldsymbol{q}^n + \boldsymbol{q}^{n-1} = \Delta t^2 M^{-1} f(\boldsymbol{q}^{n+1}), \tag{4.64}$$

which is exactly Newton's second law discretized using the standard second order centred finite difference scheme. This equation, if treated directly, requires the solution of a non-linear equation to obtain the positions of the nodes at the next step. As an alternative, we convert this equation into an equivalent optimization problem. We first define the state vectors $\boldsymbol{x} = \boldsymbol{q}^{n+1}$ and $\tilde{\boldsymbol{x}} = 2\boldsymbol{q}^n - \boldsymbol{q}^{n-1} + \boldsymbol{F}$. Note that the external forces are no longer lumped with the internal forces. This choice corresponds to the assumption that the body forces are constant in the interval $[t_n, t_{n+1}]$. We then rewrite (4.64) as

$$M(\boldsymbol{x} - \tilde{\boldsymbol{x}}) = \Delta t^2 \frac{\partial V(\boldsymbol{q})}{\partial \boldsymbol{q}}. \tag{4.65}$$

We then note that the solutions to the above equation are precisely the critical points of the equation

$$g(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \tilde{\boldsymbol{x}})^T M(\boldsymbol{x} - \tilde{\boldsymbol{x}}) + \Delta t^2 V(\boldsymbol{x}). \tag{4.66}$$

The optimization problem $\min_{\boldsymbol{x}} g(\boldsymbol{x})$ is referred to as the optimization implicit Euler's method [31]. Equivalently, this method can be rewritten to give the node positions and velocities as

$$\boldsymbol{x}^{n+1} = \text{argmin}_{\boldsymbol{x}} \left(\frac{1}{2\Delta t^2} ||M^{\frac{1}{2}}(\boldsymbol{x} - \tilde{\boldsymbol{x}}))||^2 + V(\boldsymbol{x}) \right),$$

$$\boldsymbol{v}^{n+1} = \frac{\boldsymbol{x}^{n+1} - \boldsymbol{x}^n}{\Delta t}.$$

The advantage of this optimization approach is that we can achieve a reasonable solution to this optimization problem in few iterations of standard optimization approaches such as the block coordinate descent method proposed in [31] or the method to be discussed shortly.

The result is that we can use implicit methods to achieve numerically stable solutions to this problem while avoiding the high overhead associated with using more typical implicit methods, where iterative methods must be used to achieve convergence.

While the paper [31] demonstrates the use of projective dynamics based approaches to solve this problem, we consider a more recent and general method presented in [38] which makes use of the Alternating Direction Method of Multipliers (ADMM) approach to solve this optimization problem. This work demonstrates the performance benefits of ADMM over the standard methods for optimization implicit Euler and is also more generalizable in the sense that it would be fairly easy to add alternative constitutive models to the MSS such as those shown in Section 4.3.1. ADMM is designed to solve linearly constrained optimization problems of the form

$$\min_{\boldsymbol{x},\boldsymbol{z}} g(\boldsymbol{x}) + h(\boldsymbol{z}),$$
$$\text{subject to} \quad A\boldsymbol{x} + B\boldsymbol{z} = \boldsymbol{c}.$$

It does this by introducing a dual variable $\boldsymbol{u}$ and applying the update rules

$$\boldsymbol{x}^{n+1} = \text{argmin}_{\boldsymbol{x}} \left(g(\boldsymbol{x}) + \frac{\rho}{2} ||A\boldsymbol{x} + B\boldsymbol{z}^n - \boldsymbol{c} + \boldsymbol{u}^n||^2 \right), \tag{4.67}$$

$$\boldsymbol{z}^{n+1} = \text{argmin}_{\boldsymbol{z}} \left(h(\boldsymbol{z}) + \frac{\rho}{2} ||A\boldsymbol{x}^{n+1} + B\boldsymbol{z} - \boldsymbol{c} + \boldsymbol{u}^n||^2 \right), \tag{4.68}$$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + (A\boldsymbol{x}^{n+1} + B\boldsymbol{z}^{n+1} - \boldsymbol{c}). \tag{4.69}$$

Our problem can be re-written in the form required by ADMM by setting

$$\begin{aligned} g(\boldsymbol{x}) &= \frac{1}{2\Delta t^2} ||M^{\frac{1}{2}}(\boldsymbol{x} - \hat{\boldsymbol{x}})||^2, \\ h(\boldsymbol{z}) &= V(\boldsymbol{z}), \\ A &= WD, \\ B &= -W, \\ \boldsymbol{c} &= 0. \end{aligned}$$

where W is a user-defined diagonal matrix which affects convergence speed. After setting $\hat{\boldsymbol{u}} = W^{-1}\boldsymbol{u}$ and absorbing ρ into W, $W \leftarrow \sqrt{\rho}W$, the ADMM update rules become

$$(M + \Delta t^2 D^T W^T W D)\boldsymbol{x}^{n+1} = M\hat{\mathbf{x}} + \Delta t^2 D^T W^T W(\boldsymbol{z}^n - \hat{\boldsymbol{u}}^n), \tag{4.70}$$

$$\boldsymbol{z}^{n+1} = \text{argmin}_{\boldsymbol{z}} \left(V(\boldsymbol{z}) + \frac{1}{2} ||W(D\boldsymbol{x}^{n+1} - z + \hat{\boldsymbol{u}}^n)||^2 \right), \tag{4.71}$$

$$\hat{\boldsymbol{u}}^{n+1} = \hat{\boldsymbol{u}}^n + D\boldsymbol{x}^{n+1} - \boldsymbol{z}^{n+1}. \tag{4.72}$$

Note that initially we have $\boldsymbol{z}^0 = D\boldsymbol{x}^0$ and $\hat{\boldsymbol{u}}^0 = \boldsymbol{0}$. The first, x update step (4.70) is referred to as the global step and is independent of the specific energy function used. The matrix in step one is symmetric positive definite, and therefore can be pre-factored using a sparse Cholesky factorization so that the linear system can be solved efficiently on each step.

As noted in [38], the standard Hookean potential energy $V_i(\boldsymbol{q})$ can be re-written in terms of an optimization problem,

Lemma 1 *For each* $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathbb{R}^d$*, and* $l_0 \geq 0$*:*

$$(||\boldsymbol{x}_1 - \boldsymbol{x}_0||_2 - l_0)^2 = \min_{||\boldsymbol{d}||=l_0} ||(\boldsymbol{x}_1 - \boldsymbol{x}_0) - \boldsymbol{d}||_2^2. \tag{4.73}$$

The goal of this transformation is as a relaxation, as optimization can be done over the vectors $\boldsymbol{d}$ and then a projection imposed to enforce the resting length constraint. As noted in [38], Lemma 1 allows one to write the z update step (4.71) in the ADMM as

$$\boldsymbol{p}_i^{n+1} = \text{proj}_{||\boldsymbol{x}||_2=l_0}(D_i\boldsymbol{x}^{n+1} + \boldsymbol{u}_i^n) \tag{4.74}$$

$$\boldsymbol{z}_i^{n+1} = \frac{E\boldsymbol{p}_i^{n+1} + w_i^2(\boldsymbol{x}^{n+1} + \boldsymbol{u}_i^n)}{E + w_i^2}. \tag{4.75}$$

This means that the optimization step (4.71) of the ADMM algorithm can be done very efficiently in the case of Hookean MSS's, with the optimal provisional edge node displacement vectors $\boldsymbol{z}_i^{n+1}$ being obtained in constant time.

We find that this method converges within just a few inexpensive iterations at each time level. An advantage of this approach is that if the time

Chapter 5

FSI Procedure

In this chapter, we present the main contributions of this thesis, namely, the handling of simulation scenarios where a incompressible fluid is transporting multiple immersed solid objects. In Section 5.1, we will discuss how to integrate a solid model based on MSS within a standard CFD solver. Then in Section 5.2, we will propose an efficient method for collision detection when there are multiple objects.

5.1 Fluid-Structure Interaction

In the previous two sections, we have detailed the numerical methods and algorithms that we use for computing the two constituent components to our many object simulations, namely the fluid model via the solution to the Navier-Stokes equations and the deformable solid model via MSS's. The essence of fluid-structure interaction is to integrate the fluid model and solid model together. The challenge is how one can effectively and accurately pass the information from the fluid model to the solid model and vice versa. In this section, we detail our approach for coupling the fluid and solid models.

For our model, to avoid the iterative methods that are typically required for the immersed boundary method [40], we make use of the weak coupling approach described in [7]. Specifically, with the weak coupling approach, we couple the fluid and solid together using a two step process. First, using the current velocity and pressure values on the surface of the interface, we calculate the hydrodynamics stress acting on the surface of the solids. Secondly, we apply this stress to the object and integrate the MSS's to the next time level to find the appropriate node positions and velocities using the methods discussed in the previous section.

The hydrodynamic stresses acting on the surface point of an object are given by the total force

$$F_{hydro} = \int_S \boldsymbol{\tau} \cdot \boldsymbol{n} dA + \boldsymbol{g}, \tag{5.1}$$

where $\boldsymbol{n}$ is the outward-facing normal vector for the surface S, $\boldsymbol{g}$ is the net gravitational force acting on the object and $\boldsymbol{\tau}$ is the hydrodynamic stress tensor [18] given by

$$\boldsymbol{\tau} = -pI + \mu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T), \tag{5.2}$$

where p is the fluid pressure, μ is the fluid viscosity, and $\boldsymbol{u}$ is the fluid velocity. To compute the hydrodynamic stress for each node of an MSS, we apply the trapezoidal quadrature rule to compute the net force acting on each face (line segment or triangular element in two and three dimensions respectively). The total force acting on each node is then the sum of the force contributions of each face to which the point belongs. The parameters required for the stress tensor (5.2) are computed based on the available information at each node, using the solution information in the nearest fluid cell normal direction of the boundary node. These are then applied as the nodal boundary force in the MSS integration scheme. This technique is implemented simply by using the level set function to mark the fluid cells in the staggered grid which covers the interface between the object surface and the fluid. We then compute the stress tensor in each of these cells, which are then used to compute the stresses acting on the nearest boundary node of the MSS to this cell.

Once these boundary forces have been obtained, we can integrate the MSS for the time step to obtain the updated node positions and velocities. We then apply the weak coupling strategy to tie the structural velocities to the fluid model. To do this, we impose the solid wall boundary condition

$$\boldsymbol{u}_\Gamma = \boldsymbol{u}_{\text{obj}}. \tag{5.3}$$

For the solid wall boundary condition above, we use extrapolation from the fluid domain to set the velocity values at the interface, as is visualized in Figure 5.1. To determine the object velocity, we project onto the surface of the MSS using the fast projection algorithm detailed in [22]. This allows us to quickly perform linear interpolation on the velocity of all the nodes of the nearest face of the MSS.

We must now determine what the pressure must be on the fluid-solid interface. To do this, we extrapolate the pressure from the nearby node to the interface by using the ghost cell technique detailed in [7]. This sets the pressure values "cynically" to ensure that the

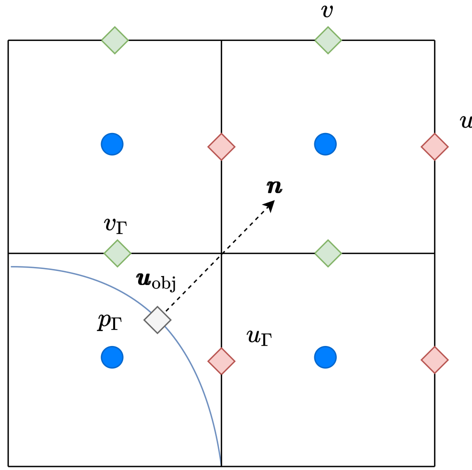

Figure 5.1: Determining the solid wall boundary condition on the staggered grid. Three dimensional case works analogously.

fluid will be incompressible near the object boundary. For a given interface cell with solid velocity $\boldsymbol{u}_\Gamma$ normal velocity $\boldsymbol{u}$, we set the pressure value to

$$\nabla p \cdot \boldsymbol{n} = \frac{1}{\Delta t}(\boldsymbol{u} - \boldsymbol{u}_\Gamma) \cdot \boldsymbol{n}, \tag{5.4}$$

which is derived through substitution into the continuity equation. This Neumann boundary condition corresponds to the projection method seen earlier to impose incompressibility by specifying the normal component of the pressure gradient appropriately.

5.1.1 Communication between the MSS's and the Implicit Surface Representation

For the solid deformation and FSI in our simulation, we have two very distinct model components that must communicate with each other. Firstly, we use a level set function defined on a fixed Euclidean grid to communicate to the fluid solver where regions containing solid interfaces are located, affecting how the fluid solver must set its boundary conditions. Secondly, we have MSS's which are Lagrangian particle systems undergoing constant deformation, which must be aware of each of the hydrodynamic and collision forces acting on their surface. For the simulation to proceed correctly, the level set function must accurately capture the boundaries of the MSS objects. To regenerate the level

set function to conform with the object on each time step would be prohibitively expensive. To avoid this, we instead make use of the level set interface motion method described in (4.2.1) to deform the level set function in response to the MSS deformations, keeping the zero-isocontour aligned with the MSS boundary.

At each interface cell we have a value $\boldsymbol{u}_{\text{obj}}$ obtained using the method described in the previous section, which is used by the fluid solver to set the solid wall boundary conditions. These velocities indicate how the object will be moving on the next step. To keep the level set function aligned with the object boundaries, and to avoid the costly operation of regenerating the conforming level set function on each time step, we must move the level set according to these velocities. To do this, we use the method detailed in [1], which details a simple augmentation to the FMM in Section 4.2.2 which can be used to extrapolate a quantity F from the isocontour such that it is constant in the normal direction, i.e., it satisfies the equation

$$\nabla F \cdot \boldsymbol{n} = 0, \tag{5.5}$$

where $\boldsymbol{n} = \frac{\nabla \phi}{||\nabla \phi||_2}$ is the outward normal vector.

We use this method to extrapolate each velocity component normal to the boundary. The resulting velocity field will move the level set function such that it captures the MSS object boundaries accurately. A velocity field of this kind is visualized in Figure 5.2, where we have extrapolated the u and v velocity components of a circle undergoing deformation. An additional advantage of this method is that the signed distance property of the resultant level set function is preserved. To see this second part, we first note that once we have extrapolated each velocity component, our velocity field satisfies the condition

$$\nabla \phi^T \nabla \boldsymbol{u}_{extrap}^T = 0, \tag{5.6}$$

which is obtained by considering that each velocity component satisfies (5.5) and is simply obtained by re-writing this set of equations in matrix form. We then consider how well the Eikonal equation (4.31) is satisfied when evolving the curve with the extrapolated velocity field $\boldsymbol{u}_{extrap}$ using equation (4.33),

$$\frac{d}{dt}||\nabla \phi||^2 = \frac{d}{dt}\nabla \phi \cdot \nabla \phi \tag{5.7}$$

$$= 2\nabla \phi \cdot \left(\frac{d}{dt}\nabla \phi\right) \tag{5.8}$$

$$= -2\Big(\nabla \phi^T \nabla \boldsymbol{u}_{extrap}^T \nabla \phi + \nabla \phi^T \boldsymbol{u}_{extrap} \cdot \nabla(\nabla \phi)\Big), \tag{5.9}$$

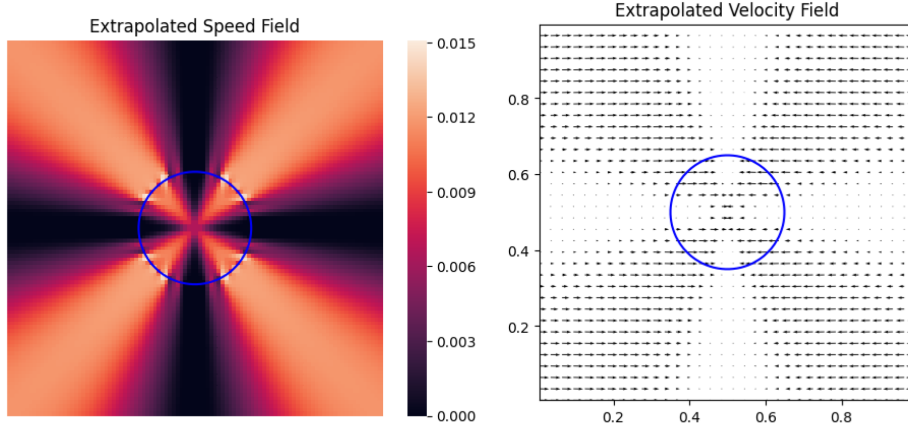

Figure 5.2: Extrapolated velocity and speed fields for a circle undergoing deformation.

where the second equality is obtained by differentiating the level set transport equation (4.33), and $\nabla(\nabla\phi)$ is the Hessian matrix of ϕ.

Using (5.6), the first term above clearly vanishes due to how our velocity field is constructed. For the second term, we make the observation from direct calculation that

$$\nabla\phi^T \boldsymbol{u}_{extrap} \cdot \nabla(\nabla\phi) = \boldsymbol{u}_{extrap} \cdot (\nabla||\nabla\phi||_2)||\nabla\phi||_2 \tag{5.10}$$

Therefore, as $||\nabla\phi||_2 = 1$ initially, the second term will vanish. Therefore $\frac{d}{dt}||\nabla\phi||^2 = 0$ and hence $||\nabla\phi|| = 1$ at all points in time when using a velocity field satisfying (5.6). This is important in our simulation, as it means that we can avoid having to regenerate the signed distance function as frequently as we would have to otherwise, while maintaining the signed distance property which is vital to our collision handling algorithm detailed in the next section.

While this method does work well in practice, the level set function can become misaligned from the MSS boundary at points. To check for this, for each boundary node we compute $\phi(\boldsymbol{x})$ using interpolation, and if this value becomes too far from 0 we regenerate the level set function using the methods described in [7]. In general, this does not happen frequently and has a small computational overhead relative to the overall cost of the simulation.

5.2 Collision Handling

In the previous section, we have detailed how our solver approaches the of fluid-solid interaction in a way which is amenable to the many object scenario. The final component of the model which must be addressed is solid-solid interaction. This is an important consideration, as our choice of using a single signed distance function to communicate the location of the object boundaries enforces constraints on how close objects can be to each other without compromising the zero-isocontour. Therefore, a key component of our solid collision approach must be the avoidance of the intersection of the isocontours belonging to separate solid objects. Additionally, our approach must deal with two complicating factors which are classically difficult: our objects are deformable and can be non-convex. This is in direct opposition to the standard assumptions used in the standard rigid body collision algorithms such as those presented in [49].

While the use of signed distance functions in the use of collision detection has been done many times previously [35, 12, 27, 27, 13]. The most similar approach to the one we take is the method introduced in the recent paper [35] for the case of colliding rigid bodies in particulate flows. This method uses probes projected from the normals of the objects to detect collisions between the objects and then uses the signed distance function to obtain information required to compute the rigid body collision response in each of the objects. Other methods include those in [52] and [24], which are particle-based methods, which allow interpenetration of the deformable objects and then use simple repulsive forces to separate them based on the depth of the overlap.

For the rigid body objects, the collision forces are well known and can be computed very accurately [49]. In the deformable case, the situation is far more complicated and is largely an area of active research. The majority of these methods in the context of finite elements augment the constitutive equations of the colliding solids with "optimality conditions", which ensure non-collision in the case of an optimal solution, the Karush–Kuhn–Tucker (KKT) conditions [28]. The KKT conditions specify *i)* object interpenetration can not occur *ii)* the directions of the collision forces and *iii)* the conditions under which collision forces are applied (e.g., forces are to be applied iff the distance between the objects is 0 at a given point). A recent example of this in a collision framework somewhat similar to our own in the context of FE is [27]. While clearly desirable, the inclusion of these KKT conditions into the material equations of motion adds an additional non-linearity to the problem which can dramatically degrade performance. At the end of this section, we argue that our method will satisfy a reasonable facsimile to the KKT conditions which avoids a good portion of the computational overhead.

The collision plane between two objects in collision at a point is the hyperplane which

intersects each of these objects at exactly one point. In the case of convex polyhedron, the existence of such a hyperplane is guaranteed by the hyperplane separation theorem. This can be seen as one of the reasons that collision detection for convex options is more computationally manageable in general, as the face segments on the boundaries of the polygonalized object can be used as candidate collision planes. A key point here is that for the case of non-convex objects, we may have multiple collisions between an object at any point, and also, the application of the collision force may easily result in another collision! We therefore design our algorithm around *a)* detecting an arbitrary number of collisions between an arbitrary number of objects *b)* ensuring collisions are unlikely to result in a collision force which causes another collision.

The first component of a collision handling algorithm is the detection of the collision and the localization of the colliding points. To perform the collision detection part, we make use of the fact that our algorithm thus far maintains a signed distance function to keep track of the surface of the objects. We say that a collision will occur when any two points on separate objects are within a collision distance δ_{col}, in practice chosen to be a small multiple of the grid spacing. If we consider the case of two objects, A and B, we can consider the domain to be divided into two regions, the subset of points that are closest to object A, and those which are closest to object B. The dividing line between these two regions is the medial axis of the exterior region Ω_t^f, here denoted by the set Φ_t. This scenario is visualized in Figure 5.3. On the boundary separating these two regions, the value of the signed distance function indicates the distance between the two objects. with a collision being possible if $\phi(\boldsymbol{x}) < \frac{\delta_{\text{col}}}{2}$ on this boundary.

In terms of signed distance functions, Φ_t is the set of points that have no unique closest point to the isocontour, e.g., the result of the closest point formula (4.32) is undefined. In our algorithm, we use the medial axis to build a representation for the collision surface, a generalization of the collision plane for the non-convex case. The point of this is to build a quite rough approximation of where collisions of the objects may take place, and for the purposes of avoiding interpenetration of the objects, considering this to be a collision and respond accordingly.

While approximation of the medial axis of a signed distance function is difficult to do accurately, a sufficiently accurate approximation of the medial axis is obtainable using the methods for level sets we have already discussed. In particular, we run the FMM in the exterior domain of the level set, terminating the march when the value of ϕ exceeds $\frac{\delta_{\text{col}}}{2}$, meaning that the FMM only needs to run within a small band of cells $\frac{\delta_{\text{col}}}{2}$ from the boundary of each object. When the algorithm discovers a grid point that has already been discovered by the march extending from the other object, this is labelled as a medial axis cell and the boundary of this cell is used to approximate the location of a medial axis point.

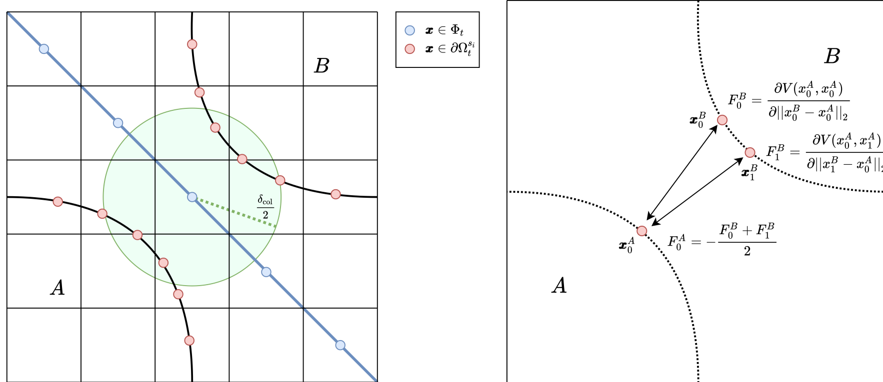

Figure 5.3: (Left) Collision detection process using radius search from medial axis points of the current object boundaries, Φ_t. (Right) Computing the spring-based repulsive force based on the proximity of a node on object A to nodes in object B at the time of collision detection.

This gives a rough approximation that is sufficiently accurate for our application, with the error being bounded by the grid size. A similar algorithm of this kind was presented in [46], which combines the FMM algorithm of the kind we implement for our medial axis discovery with some refinements for applications where more accuracy is needed.

Following this process, we have a set of medial axis points which indicate the boundaries between objects where collisions are possible. Following this step, we must now identify the MSS's that are in contact and localize the contact onto the surface of each MSS. The first step is done by maintaining information from the FMM procedure to indicate medial axis points which correspond to a collision in a subset of the MSS's. This is done by propagating information about which object each step of the FMM stems from, that is, for each grid cell using information gained from the boundary of object A, we label this grid cell as belonging to the subset of domain points which are closest to object A. An example of this in a scenario with many objects is shown in Figure 5.4. Once we have determined which objects are in contact, we store the boundary nodes of the colliding MSS's in a $k-d$ tree using the nanoflann library [5] to enable efficient lookup of the nodes that are engaged in the current collision. From each medial axis point, we perform a radius search about each medial axis point with radius δ_{col} using the built $k-d$ tree to determine the subset of nodes on each MSS which are in collision with. These are recorded for the next step.

Once we have built a set of colliding nodes, we must then determine the collision force to apply to each of them. However, to ensure non-penetration we must cancel the momentum

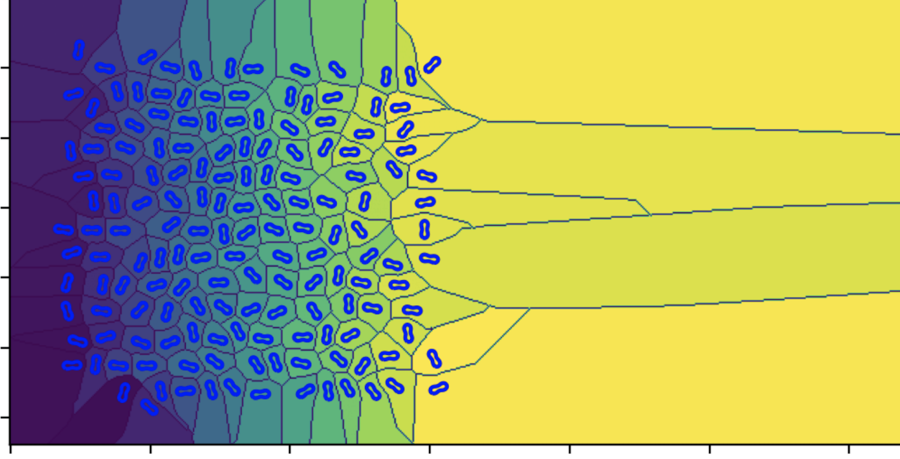

Figure 5.4: Graphical representation of regions of the domain identified as being closest to a given object, as labelled during our modified FMM.

of the colliding boundary nodes by cancelling out the internal forces acting on it before applying the collision force. To do this, we compute the internal spring forces for the springs connected to the boundary nodes,

$$\boldsymbol{F}^n_{\text{internal}}(\boldsymbol{x}_\Gamma) = \sum_{i\,\text{connected}} E\frac{||\boldsymbol{x}_\Gamma - \boldsymbol{x}_i|| - l_0}{\epsilon_i}(\boldsymbol{x}_\Gamma - \boldsymbol{x}_i), \tag{5.11}$$

where $\boldsymbol{x}_\Gamma$ is a given MSS boundary node undergoing collision and ϵ_i is the current strain of the ith spring connected to this node, and l_0 is the resting length of this spring. When the object has undergone a large amount of deformation, this step helps to ensure that the internal forces acting on boundary nodes will not exceed the external forces due to the collision. This may cause the objects to deform into each other, resulting in interpenetration. This can be seen as a correction based on applying Euler's method to only this node and cancelling out the resultant stress. We must also add sufficient force such that this node will be stationary following this time step, and before the collision response force is added. This is calculated as

$$\boldsymbol{F}^n_{\text{stationary}}(\boldsymbol{x}_\Gamma) = \frac{m}{\Delta t_n}\boldsymbol{v}^n(\boldsymbol{x}_\Gamma). \tag{5.12}$$

In summary, we calculate the momentum and velocity cancellation as the force

$$\boldsymbol{F}^n_{\text{cancel}}(\boldsymbol{x}_\Gamma) = \boldsymbol{F}^n_{\text{internal}}(\boldsymbol{x}_\Gamma) - \boldsymbol{F}^n_{\text{stationary}(\boldsymbol{x}_\Gamma)}. \tag{5.13}$$

In our model, we make use of simple spring-based forces based on the proximity of the objects to each other. We specify the resting length of the spring applying the collision force as $l_{\text{col}} = 2\delta_{\text{col}}$, as we find this works well in practice. This stiffness of the collision spring, E_{col}, can also be varied to give more or less repulsive force. For each collision detected, we lump the nodes within this radius as being colliding.

$$\boldsymbol{F}_{\text{repulse}}(\boldsymbol{x}_{\Gamma_A}) = \frac{1}{n_{\text{c}}} \sum_{x_{\Gamma_B}} E_{\text{col}} \frac{||\boldsymbol{x}_{\Gamma_A} - \boldsymbol{x}_{\Gamma_B}|| - \delta_{\text{col}}}{||\boldsymbol{x}_i - \boldsymbol{x}_j||} (\boldsymbol{x}_{\Gamma_A} - \boldsymbol{x}_{\Gamma_B}), \tag{5.14}$$

The final collision force acting on a boundary node $\boldsymbol{x}_\Gamma$ is calculated as

$$\boldsymbol{F}^n_{col}(\boldsymbol{x}_\Gamma) = \boldsymbol{F}^n_{\text{repulse}}(\boldsymbol{x}_\Gamma) + \boldsymbol{F}^n_{\text{cancel}}(\boldsymbol{x}_\Gamma). \tag{5.15}$$

Using this method we have observed good experimental performance in terms of creating realistic physical simulations with object-to-object collisions. In terms of the mathematical principles of this method, we consider how our method relates to the KKT conditions. In terms of the conditions listed earlier, *i)* object interpenetration can not occur, as we apply repulsive forces to ensure objects can not drift into regions occupied by other objects; *ii)* we specify the direction of the collision force applied for each detected collision on an MSS boundary node by applying a repulsive force as the net force of each node is it detected as being in contact with through the equation (5.14); *iii)* we specify that a collision occurs iff the distance between any two nodes on nearby objects are within approximately δ_{col} from each other.

To end this section, we show some preliminary results demonstrating the efficacy of the algorithm for some small-scale collision problems. First we consider the interaction of two circular objects in two dimensions approaching each other and colliding in Figure 5.5. A value for δ_{col} which we find to work well, in general, is $3h$, where h is the grid spacing. We find that this value is effective in general. While this does not look physically realistic in these small examples, for more realistic examples in our simulation, the grid spacing is far more fine.

For our second example, we use several convex objects moving at various initial trajectories in Figure 5.5, where we see that they deform and then are repulsed in the way that would be expected through the use of our algorithm. For our third example in Figure 5.7, we showcase our algorithm when used with a finer grid with non-convex objects. This demonstrates the physical realism that is attainable by using fine grids with this method, as well as demonstrating the generality of the code for convex objects.

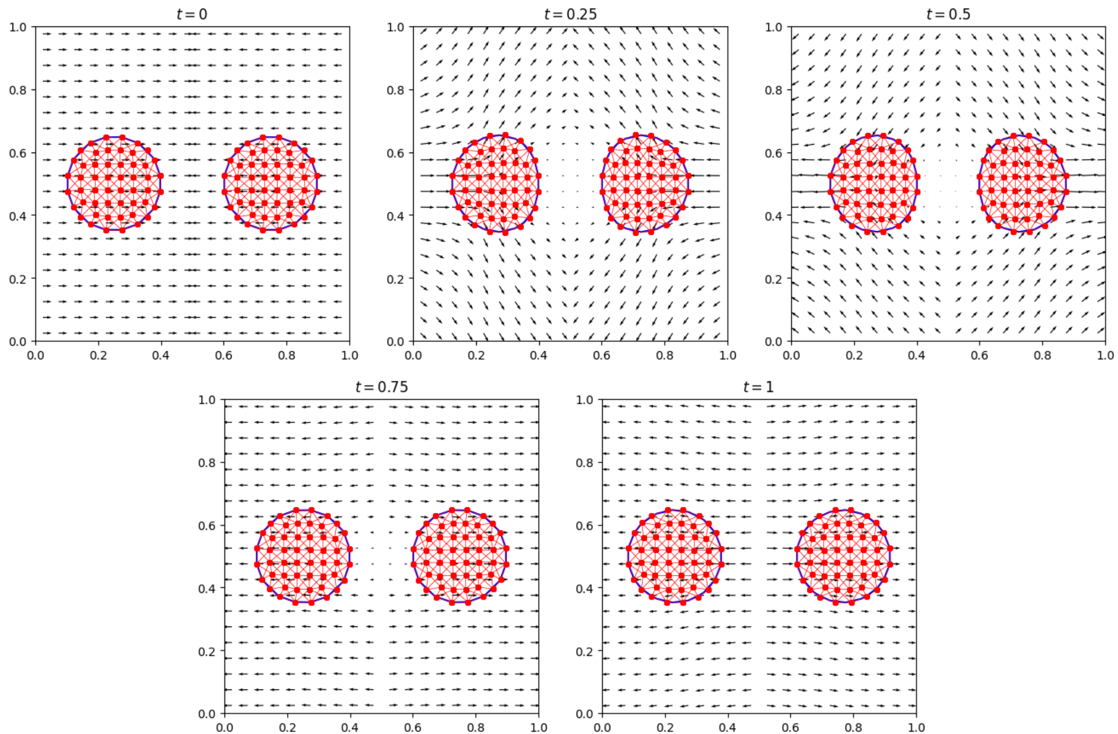

Figure 5.5: Basic collision between 2 convex objects in absence of fluid (initial object velocities chosen by hand). The vectors are the normally extrapolated velocities from the surface of the objects. The objects proceed towards each other until a collision is detected, the collision handling them applies the collision forces. This results in deformation in each of the objects, and for them to bounce away from each other. Note that some distance δ_{col} must be maintained between objects, hence the large gaps for coarse grids.

5.3 Summary of Simulation Algorithm

In this section, we give a summary of our simulation algorithm to provide clarity in how each of the model components have been brought together. We consider the voxelization of the domain Ω^d at time t, with grid cells labelled into fluid, solid and interface regions, i.e., elements of $\Omega^f_{t_n}$, $\Omega^s_{t_n}$, and Γ_{t_n} respectively. We are given the initial fluid velocities $\boldsymbol{u}_i^n$ for each grid cell i, with each component of the velocity arranged at the appropriate place on the staggered grid. The fluid component of the model is then updated to the next time level $t_n \mapsto t_{n+1}$ using the following procedure:

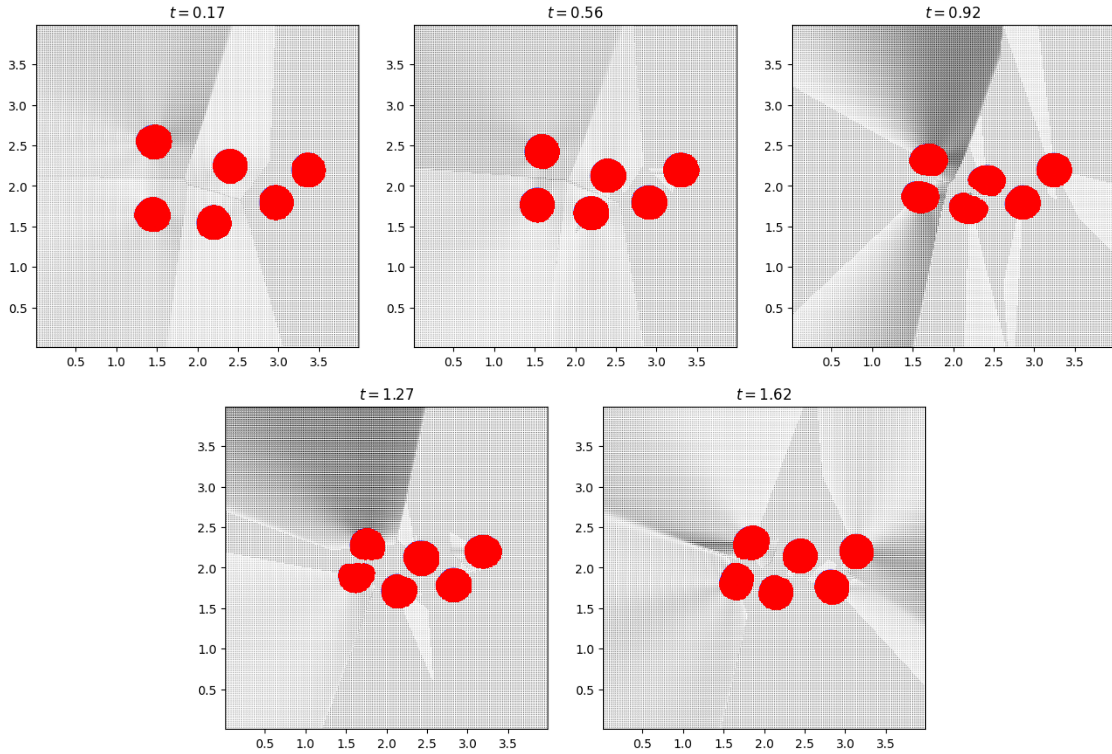

Figure 5.6: Collision between convex objects moving at varying initial velocities to induce collisions. We see here the capabilities of the collision detection algorithm for managing collisions between several objects simultaneously.

1. Boundary conditions are assigned at the appropriate locations within the staggered grid including
   - the domain boundaries (as described in Section 4.1.3)
   - the object boundaries (solid wall boundary condition, Section 5.1) using the velocities extrapolated from Γ
2. The implicit-explicit solution procedure is used to obtain the values $\boldsymbol{u}^{n+1}$ and p^{n+1} satisfying the continuity equation, as described in Section 4.1.1.

Following the advance of the fluid model, we then must update the solid component of the model, as well as coordinate the two models. Additionally, the FSI algorithm is applied to facilitate structure-structure interactions. This is done in the following steps:

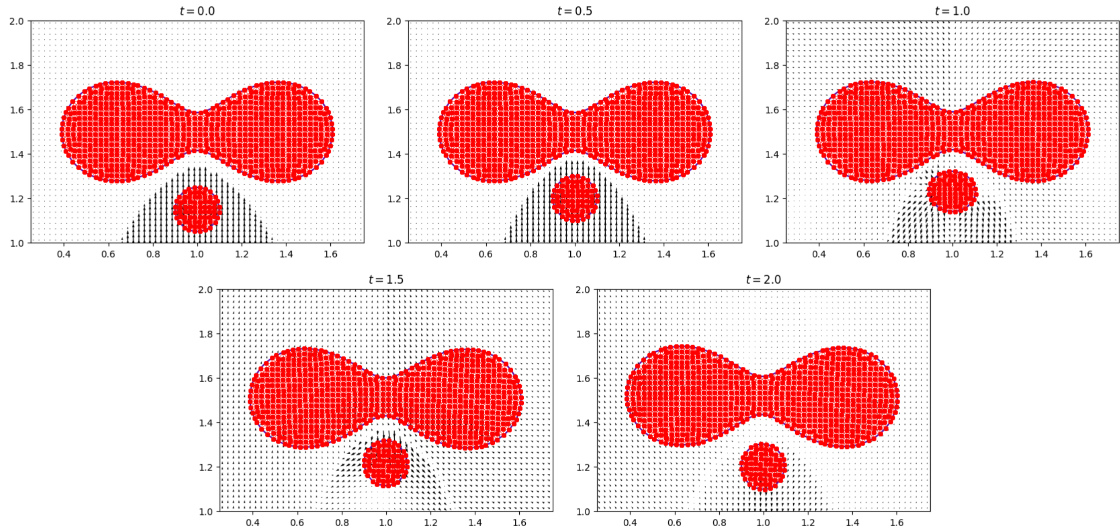

Figure 5.7: Basic collision between a convex and concave objects as a demonstration of our ability to handle non-convex object collision.

1. We compute the medial axis for the current object configurations and use our FSI procedure from Section 5.2 and use this to compute the collision stress $\boldsymbol{F}_{\text{col}}$.

2. The hydrodynamic stresses $\boldsymbol{\tau}^{n+1}$ are computed and applied to the boundaries of the object along with the body and collision forces.

3. The MSS positions and velocities are updated using the ADMM method described in Section 4.3.2.

4. The updated fluid and solid domains are obtained by evolving the level set function. The object velocities are extrapolated as described in Section 5.1.1 and then are used to evolve the level set function as described in Section 4.2.1.

5. If required, the level set function is re-initialized to a signed distance function using the FMM, or the level set function is re-built to coincide with the boundaries of the MSS.

The end result of this procedure is an incompressible velocity field $\boldsymbol{u}^{n+1}$ and pressure field p^{n+1}. The MSS is deformed appropriately based on the forces acting on it and has a new set of boundary velocities to communicate the solid wall boundary conditions to the fluid solver. Additionally, we have a deformed domain with updated configurations

$\Omega^f_{t_{n+1}}$ and $\Omega^s_{t_{n+1}}$ for the fluids and solids respectively, as indicated by the level set function, informing the fluid solver as to the boundary configuration for the next time step. Care is taken to prevent the intersection of objects during the simulation, and a fairly fine grid may be necessary to prevent intersection from occurring in a physically realistic way, as well as to capture finer scale deformations of the objects.

Chapter 6

Results

In this chapter, we apply the method for many-object FSI which we have formulated over the previous Chapters to some model scenarios. The purposes of these simulations are two-fold: *i)* to demonstrate that our method can produce physically realistic simulations of our problem class *ii)* to demonstrate the scalability of the simulation method. We begin with describing the scenarios we are working with for the two-dimensional case, and proceed to analyze the performance of our method in these cases.

In the following sections, quantitative and qualitative results are presented, with an accompanying analysis of how the method performs in each scenario.

6.1 Scenario 1: Lid Driven Cavity

The purpose of this example is to demonstrate some of the core functionalities of our solver in a fairly simple scenario. In this example, two moving non-convex objects above and below a set of convex objects are seeded with initial velocity $\boldsymbol{u}_0 = \mathbf{0}$ and propelled by a lid-driven cavity flow with velocity $\boldsymbol{u}(x, y_{\text{lim}}) = \begin{bmatrix} u_{\text{lid}} & 0 \end{bmatrix}^T$. We expect this to induce collisions between the seeded convex and non-convex objects as the lid-driven flow causes the non-convex objects to begin to rotate and drift upwards. The diagram for this scenario is shown in Figure 6.1. The orange zone in this diagram indicates where seeded objects are placed in each realization of this scenario, blue areas indicate areas initially occupied by fluids. The white locations are deformable non-convex objects which will be included in each realization of this scenario. Two examples running with varying numbers of objects and grid sizes are shown in Figure 6.2. In this figure, we plot the state of the simulation at

each point in time, plotting the objects in red and plotting the fluid velocities $\boldsymbol{v}$ as black vectors whose length indicates the magnitude of the velocities. Note that for larger grid sizes the shape of the individual vectors is lost, however, the shade of a given point indicates the magnitude of the velocities at this point and the direction of the fluid flow is intuitable. The purpose of this scenario is to demonstrate collision between convex and non-convex objects, particularly objects of drastically different sizes, as well as simply demonstrating the ability of our method at producing visually realistic results.

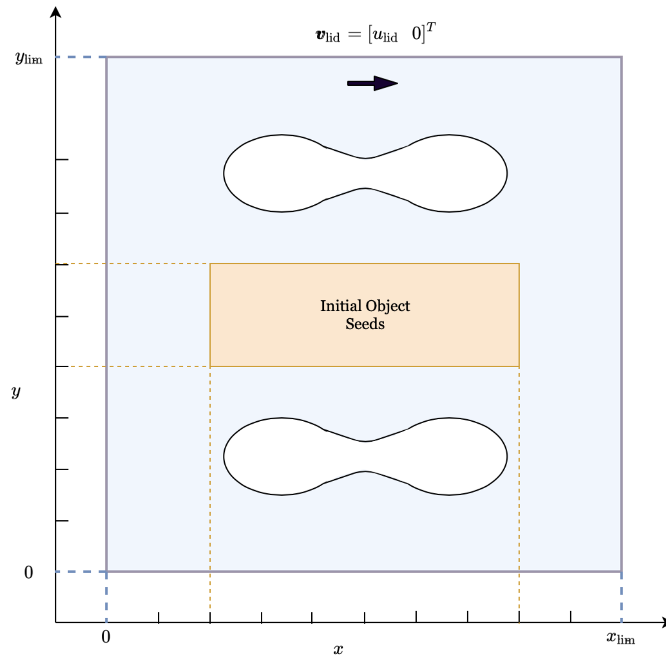

Figure 6.1: Scenario 1 diagram.

In this scenario, as we proceed in time, we see that the topmost object is propelled upwards by a negative pressure differential at the top-left corner while being rotated by the positive pressure at the top-right corner. In the more dense example shown in Figure 6.2, the boundary conditions produce a large swirling velocity field in the top-right corner which eventually propels the smaller objects upwards once it grows large enough. This, combined with the motion of the non-convex objects induces a handful of collisions with the small circular objects, which the solver can handle without compromising the iso-contour of the level set function.

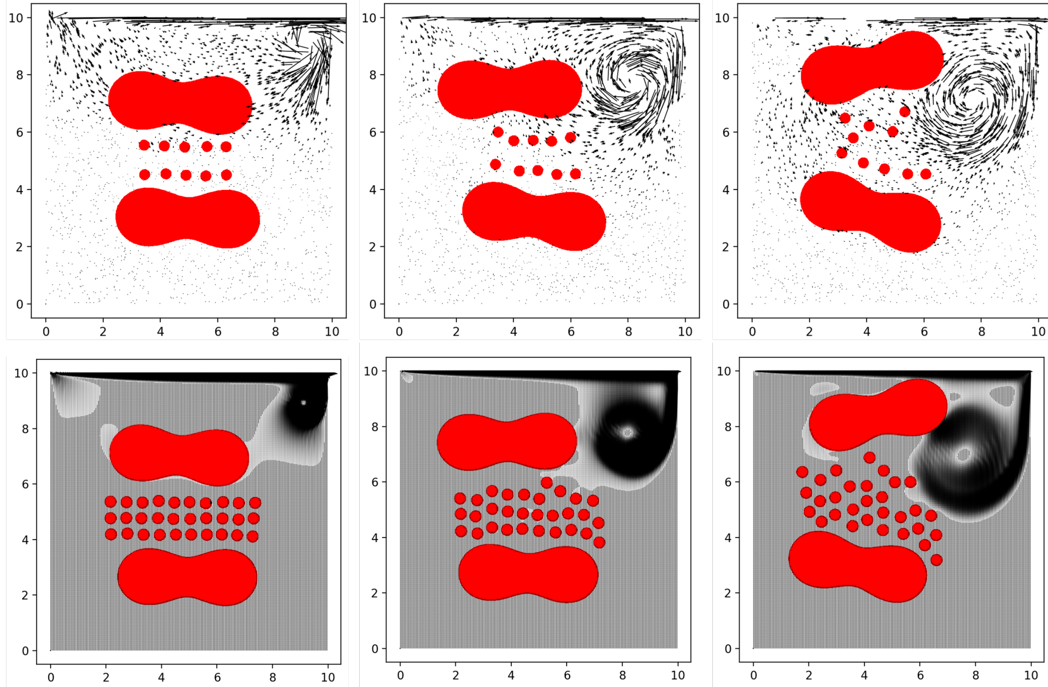

Figure 6.2: Scenario 1 realization for varying numbers of objects and grid size. (Top) 10 seeded circular objects on a grid of size $\boldsymbol{N} = (80, 80)$. (Bottom) 30 seeded circular objects on a grid of size $\boldsymbol{N} = (160, 160)$. in each case the solution is given at $t = 5, 10, 15$.

6.2 Scenario 2: Object Tumbling

In this example, we have several objects with identical mass and density seeded with an identical initial velocity. At the northern edge of the domain, we have an inflow condition specifying in a fluid with $Re = 1000$. These objects tumble together until reaching a stationary object which is placed to agitate the falling objects and demonstrate the collision detection. This is similar to the example posed in the paper [35], which uses a level-set method somewhat similar to the method we propose. In this related work, the authors used just two objects in a two-dimensional scenario to demonstrate their collision algorithm, which uses an approximation of standard rigid-body dynamics to handle collisions between objects. Thus the performance of our method with this one is not directly comparable. Additionally, the authors did not add a stationary object to disrupt the flow of the tumbling objects. The expected behaviour of this scenario is that objects at a lower height will collide with the object first, causing a cascade of collisions with the objects above them, while

the objects that don't hit the object head-on slide by, propelled by the gravitational force and the fluid pressure from the inflow boundary condition at the top of the domain. An example of this scenario proceeding over time in two dimensions is shown in Figure 6.3, where $\boldsymbol{u}_0 = \begin{bmatrix} 0 & -\frac{1}{2} \end{bmatrix}$.

In the sparse example in Figure 6.3 we see that the middle-most objects collide first and bounce off of the stationary obstruction, causing the objects immediately above them to in sequence collide with the layer under them. The other objects begin to collide with the side of the stationary object and begin to rotate and slide off of the side of the object, as is visualized in Figure 6.5. In the dense scenario shown in Figure 6.4, we see that the middle-most objects, after the collision, begin to deform as the stress of continuous collisions are applied to their boundaries. The objects at the sides begin, particularly on the left-hand side of the stationary object, to get stuck at the edges of the objects. This is because, in the simulation logic, these become tightly packed and it takes some time for objects to make their way free, as can be seen with a handful of objects exiting the pile-up and being propelled to the bottom due to the gravitational force. This can be somewhat addressed by increasing the grid density, which in turn reduces the minimum acceptable value of the collision distance δ_{col} in the collision handing algorithm discussed in Section 5.2.

6.3 Scenario 3: Flow Through Channel

In this example, we provide some preliminary simulations in the motivating problem class for this problem, the multi-scale modelling flow of blood through a vessel. This is simpler than the other scenarios in that we embed many copies of the same object, showing the distinct "blood cell" shape, generated as a cross-section of the three dimensional shape and then rotated randomly. Note that we refer to simulated Red Blood Cells as RBC's. The diagram for this scenario is visualized in Figure 6.6, and an example for relatively few objects is given in Figures 6.7, where the initial conditions for the scenario are given as $\boldsymbol{u}_0 = \begin{bmatrix} \frac{1}{2} & 0 \end{bmatrix}$ for both the fluid and inflow and outflow boundary conditions on the left and right-hand sides of the domain, as visualized in Figure 6.6.

This scenario is a stress test for the method in that it must prevent and respond to many largely unpredictable collisions between the many objects. Additionally, this will also involve a large amount of collision between non-convex objects, where our handling of this case is a defining characteristic of our method. Note that if we simply set the initial velocities of the objects to be equal to $\boldsymbol{u}_0$, the objects will proceed through the domain with largely no interaction with each other, as there is no pressure differential which can induce the objects to move into each other. To add additional complexity and demonstrate the ability of our method for handling a large amount of collisions between these objects, we set the initial conditions of the objects to include a random small perturbation to the initial conditions as follows, $\boldsymbol{u}_0^i = \boldsymbol{u}_0 + \boldsymbol{r}^i$ where $||\boldsymbol{r}^i|| < \frac{1}{10}||\boldsymbol{u}_0||$ for object i. While this is not in practice analogous to the scenario we would consider with a full multi-scale simulation of blood flow, we consider this to be a good demonstration of the capabilities of our method for this problem class.

We begin with an example of this scenario with a relatively small number of immersed RBC-type objects in Figure 6.7. We see here that as we proceed forward in time, the simulation diverges from a relatively uniform flow of the objects into a large number of collisions between the immersed objects due to the initial small perturbations of the object velocities. Where the objects collide, we see clumps of RBCs forming towards the center of the domain, with those at the end of the domain moving relatively unhindered. One observation is that near the start of the domain, there are several slow-moving objects in collision with each other. This is an observation we see in several of our simulations, where collisions can result in fairly dramatic changes in the velocities of the objects, with two objects becoming "stuck" in place. A contributing factor to this is the fact that the velocity of the colliding objects is cancelled in the collision stress explained in Figure 5.2. These terms were designed to avoid catastrophic intersection of the object surfaces but they are quite conservative in their design. An improvement to our collision algorithm which could

mitigate this phenomenon is a modification of the $\boldsymbol{F}^n_{\text{cancel}}$ to only act to mitigate the velocity in the normal direction of the collision. The normal direction could be easily approximated using the gradient of the level set function, as is done in [35]. This modification would act to avoid this issue, without risking object intersection.

A more complex example, with smaller immersed objects is shown in Figure 6.8, with 700 objects placed in the scenario. In addition to the typical plots we have shown for our two-dimensional simulations, we give close-ups of a specific region of the simulation over time to improve clarity. In particular, we see major deformations in the objects as t increases due to collisions between neighbouring objects. In the whole of the domain, we see the objects forming clumps and rotating rapidly due to detected collisions.

This scenario has demonstrated the ability of our method in performing simulations in our target application. The next question is whether the method achieves the scalability necessary for a fully-resolved blood flow simulation with a realistic quantity of objects. This is explored in the following section, where we perform scaling experiments to deduce how well our method performs in terms of execution time.

6.4 Scaling Performance

In this section, we evaluate the scaling performance of the method on these testing scenarios. Let our simulation scenarios be defined on a numerical grid with the number of grid cells in each component direction $\boldsymbol{N} = (N_x, N_y, N_z)$ and corresponding grid spacings $\boldsymbol{h} = (h_x, h_y, h_z)$. The two most relevant scaling effects we wish to measure in our solver are the effect of adding additional objects to the simulation for a fixed grid size $\boldsymbol{h}$, which we will refer to as object scaling, and the cost of adding additional grid points to meet the required grid density for objects of a certain relative size, which we refer to as simultaneous scaling.

Note that in the case of the first scaling experiments, object scaling, where we are simply increasing the number of objects in the scenario with a fixed grid size, we ensure that the density of the objects within each scenario remains the same to keep the frequency of object collisions similar in each experiment.

To elaborate on the second scaling measure, simultaneous scaling, consider an object with its surface identified as the zero-isocontour of a signed distance function $\phi(\boldsymbol{x})$ defined on a grid of size $\boldsymbol{N}$. As was shown in Figure 4.2, our code uses an enumeration of each grid cell to distinguish between, solid, fluid, and interface cells on the staggered grid. With this in mind, the practical minimum size of an object is one whose major axis is approximately

$3\max\{\boldsymbol{h}\}$. This is visualized in Figure 6.9. Therefore, if we wish to model a scenario with a specific number of objects of a certain size, there is a minimum number of grid cells that are required to accommodate these grid cells with the level set function. Therefore, in our simultaneous scaling tests, we set up each of our test scenarios by packing in as many objects as possible with major axes of size $\alpha\max\{\boldsymbol{h}\}$, where α is a factor which depends on the objects being used, to measure how the performance changes in each scenario when we increase how fine the grid is, and simultaneously decrease the size of the objects and increase the number of immersed objects. The purpose of these experiments is therefore to evaluate the limitations of our grid-based solver in the context of accommodating a large number of objects. Note that in theory the grid of the level set function and the fluid could be separate, and interpolation could be used to communicate between the grids. This would alleviate the increase in cost due to needing a finer grid, by using a fine grid for the level set function and a coarse grid for the fluid solver. This adds complexity to the communication between the model components, and we leave this as a topic for future work.

In this section, we evaluate the scaling performance of the method on these testing scenarios. To begin with, we discuss the results of the object scaling tests, where we are increasing the number of objects for a particular fixed grid size. In Table 6.1, we present the results for each of a set of experiments for each scenario. The table is separated into three sections, one for each scenario, labelled with the scenario and the value of $\boldsymbol{N}$, the fixed grid size used for each of these object scaling experiments. In each section, we present the number of objects placed in the scenario and the total CPU time it took to reach t_{end} in seconds, as well as the average time per step.

Looking at these data in Table 6.1, as we increase the number of objects, we see an unpredictable relationship between the number of objects and the CPU time in each experiment. This is due to a confluence of factors in the performance of the solver, but the largest of these is because as more objects are embedded, more grid cells are enumerated as belonging to the structural domain Ω_t^s. Hence the Poisson equation (4.10) needs to be solved with fewer unknowns, which can offset the cost of integrating the MSS models for the additional models. This leads to the observation that when using these low-order MSS models for deformable solids, their integration essentially comes at no additional cost to solving the equivalent generic fluid flow problem with no domain obstacles. One outlying performance that we observed was in the case of Scenario 1 when 80 objects were used. In this case, it was due to a large number of collisions that resulted in collision velocities that were greater than the maximum velocity in the other experiments in this scenario. This resulted in more iterations being required to solve the Poisson equation, as these newly introduced velocities introduced in the scenarios may have made the initial guess for the

conjugate gradient in solving the Poisson equation be further than what is required, and hence requiring more iterations on average.

For the simultaneous scaling experiments, in Table 6.2, we give the number of unknowns and objects used for each experiment and the resultant total and average per-step run times required to reach t_{end} for each experiment. The α value indicates the size of the objects as a multiple of the grid size as previously discussed. In our method, we would expect the asymptotic performance of the method is somewhat bounded below by the performance of the standard numerical methods that we implemented. The cost of this is bounded by the linear solve for the Poisson equation, which when using the conjugate gradient method is typically observed to be super-quadratic in the number of unknowns N. Examining these tables, we take note of the fact that, as expected, in each case as the number of unknowns and number of objects used in the scenario increases, so too does the execution time.

To quantify how the CPU time scales with respect to N, we plot the total and average run times for each experiment and scenario in the Figures 6.10, 6.11 and 6.12. Additionally, we apply linear and quadratic least squares to each of these data sets and give the sum of squared errors (SSE) for the curves. From these visualizations and looking at the SSE values, we see that universally, the average run-time for the method is well-fit by a quadratic function in each case, as is expected. While in the total time plot, each is well fit by a quadratic except for Scenario 2. Examining this scenario, we observe that in the case in Table 6.2, in Scenario 2, for $\boldsymbol{N} = (320, 640)$, the Total Time is far higher than would be expected when comparing to the analogous results in the other two scenarios. This is due to a large number of collisions in this particular scenario resulting in greater velocities in the simulation, which in turn shrunk the CFL condition such that this scenario required far more time steps to reach t_{end}. In this we observe an issue with our proposed collision algorithm: it can introduce additional kinetic energy into the simulation which can eventually degrade performance. Examining how to mitigate this issue would be an important step in deploying this method in larger-scale simulations.

Our simultaneous scaling experiments also indicate that improvements to our method would likely be required to embed a large number of small objects. Looking at the relationship between the number of objects (note that this is a function of the parameter α and the grid size N in this scenario) and the total run time, this follows a quadratic trend as well, which can be seen from the fact that in Table 6.1, the number of objects scales approximately with the same factor that N is. Essentially, as we increase how fine the grid is by a factor, the number of objects we can embed only increases by this factor, as the minimum allowable size only decreases by this factor. The idea discussed previously where a different grid size is used for the level set function has already been discussed, but again this is a subject for future work.

Number of Objects	Total Time (s)	Average Time ($\frac{s}{\text{step}}$)
Scenario 1 with $\boldsymbol{N = (640, 640)}$		
20	12070	9.21
40	13283	10.13
80	21738	16.58
160	11918	9.09
Scenario 2 with $\boldsymbol{N = (640, 1280)}$		
20	31844	9.72
40	32842	10.02
80	29948	9.14
160	33304	10.17
Scenario 3 with $\boldsymbol{N = (1280, 640)}$		
40	8089	26.96
80	7538	24.96
160	7188	23.88
320	7474	24.76
640	6723	22.26

Table 6.1: Object scale test data for each of the test scenarios.

In summary, examining the performance of our method, we see some promise in terms of using our algorithms for scalable FSI simulation. For a fixed grid size, we observe that our method does not result in a substantial overhead over the standard methods we apply for the fluid portion of our simulation, and in that sense, scales well concerning the number of objects we choose to use. On the other hand, our choice of representation of these objects, and how we choose to use it in our code (the alternatives have already been discussed) limits the effective scaling of our solver purely in terms of the number of objects that we can include in our simulations.

6.5 Three-dimensional Results

As setting up scenarios in three dimensions is a more complex task, and the computational power required to attain the results increased substantially with the additional dimension, we provide a more limited discussion of the three-dimensional case, focusing primarily on the results for Scenario 2 presented in Section 6.2. This scenario is a good test for the generalization of the solver into three dimensions, as its easy set-up allows us to evalu-

$\boldsymbol{N}$	Number of Objects	Total Time (s)	Average Time ($\frac{s}{\text{step}}$)
Scenario 1 ($\alpha = 3$)			
40×40	2	1	< 0.01
80×80	2	2	0.02
160×160	7	40	0.16
320×320	32	689	1.22
640×640	122	11276	8.60
Scenario 2 ($\alpha = 3$)			
40×80	2	1	< 0.01
80×160	7	11	0.07
160×320	25	111	0.39
320×640	118	1729	2.39
640×1280	469	5576	21.20
Scenario 3 ($\alpha = 4$)			
80×40	3	< 1	< 0.01
160×80	18	2	0.02
320×160	72	26	0.14
640×320	312	254	0.79
1280×640	1296	2880	9.50

Table 6.2: Full scaling data for each of the test scenarios

ate many of the important components of the solver. Most importantly, it allows us to demonstrate the ability of our method to easily generalize into the three-dimensional case, as 3D analogues of each of the component algorithms summarized in Section 5.3 can be done reasonably simply using methods that are very familiar to those versed in 3D physical modelling.

An example of Scenario 2 in three-dimensions, running from the starting time of $t = 0$ to $t = 10$ with 49 immersed objects is shown in Figures 6.15 and 6.16. In these figures, we plot the state of the three-dimensional scenario at a particular fixed perspective, and also give two-dimensional slices of the level set function at fixed y values to show more clearly how the three-dimensional level set function is changing with time. Note that in these two-dimensional plots, the aspect ratio is flattened in the interest of space, making the isocontours of the spherical objects look elliptic. In the three-dimensional plot, the immersed objects in the scenario are represented by plotting the isocontours, as well as the mass particles of each MSS on the surface of the object, with each being different colours. Cone plots are used to visualize the velocity field. In this scenario, we see largely the same behaviour as in the sparse tumbling example shown in Figure 6.3. In particular, we witness the collision of the bottom-most objects, causing subsequent collisions with the objects above them. The grid is fairly coarse in this example, making the collision handling appear visually non-realistic, however, this example does demonstrate that our methods generalize well to the higher-dimensional case.

To examine whether the scaling performance of the solver is maintained in the three-dimensional case, we run the simultaneous scaling experiment which we have previously performed for the two-dimensional results. These results are presented in Table 6.3 which is identical in structure to those presenting the simultaneous scaling results in the previous section. In this table, we observe the same trends as those presented in the previous section, indicating that the observations in the simultaneous scaling analysis we made about the two-dimensional scenario are also applicable to the three-dimensional scenario. The object scaling results for this three-dimensional example are shown in Table 6.4. In this table, we see an even stronger trend pointing towards the fact that for a fixed grid size, the addition of more objects into the scenario results in lower overall execution time, likely for the same reasons discussed in the previous sections.

Finally, in the interest of making headway into the target application for this research, we provide a simple example of the three-dimensional generalization of Scenario 3, described in Section 6.3. This course example with 49 immersed, RBC-shaped objects are plotted in Figure 6.15 and 6.16. Here we see that the red blood cells follow the expected trajectory and avoid issues that could be expected, such as the intersection of the zero-isocontours of the level set function.

N	Number of Objects	Total Time (s)	Average Time ($\frac{s}{\text{step}}$)
$40 \times 40 \times 40$	2	1	0.25
$80 \times 80 \times 80$	2	30	3.75
$160 \times 160 \times 160$	10	820	45.56
$320 \times 320 \times 320$	101	25320	617.561

Table 6.3: Full scaling data for each of the test Scenarios in 3D

Number of Objects	Total Time (s)	Average Time ($\frac{s}{\text{step}}$)
40	46200	745.161
80	45383	731.984
160	43158	696.097

Table 6.4: Object scaling data for Scenario 1 in 3D, $\boldsymbol{N} = (320, 320, 320)$ run to $t_{\text{end}} = 1.5$

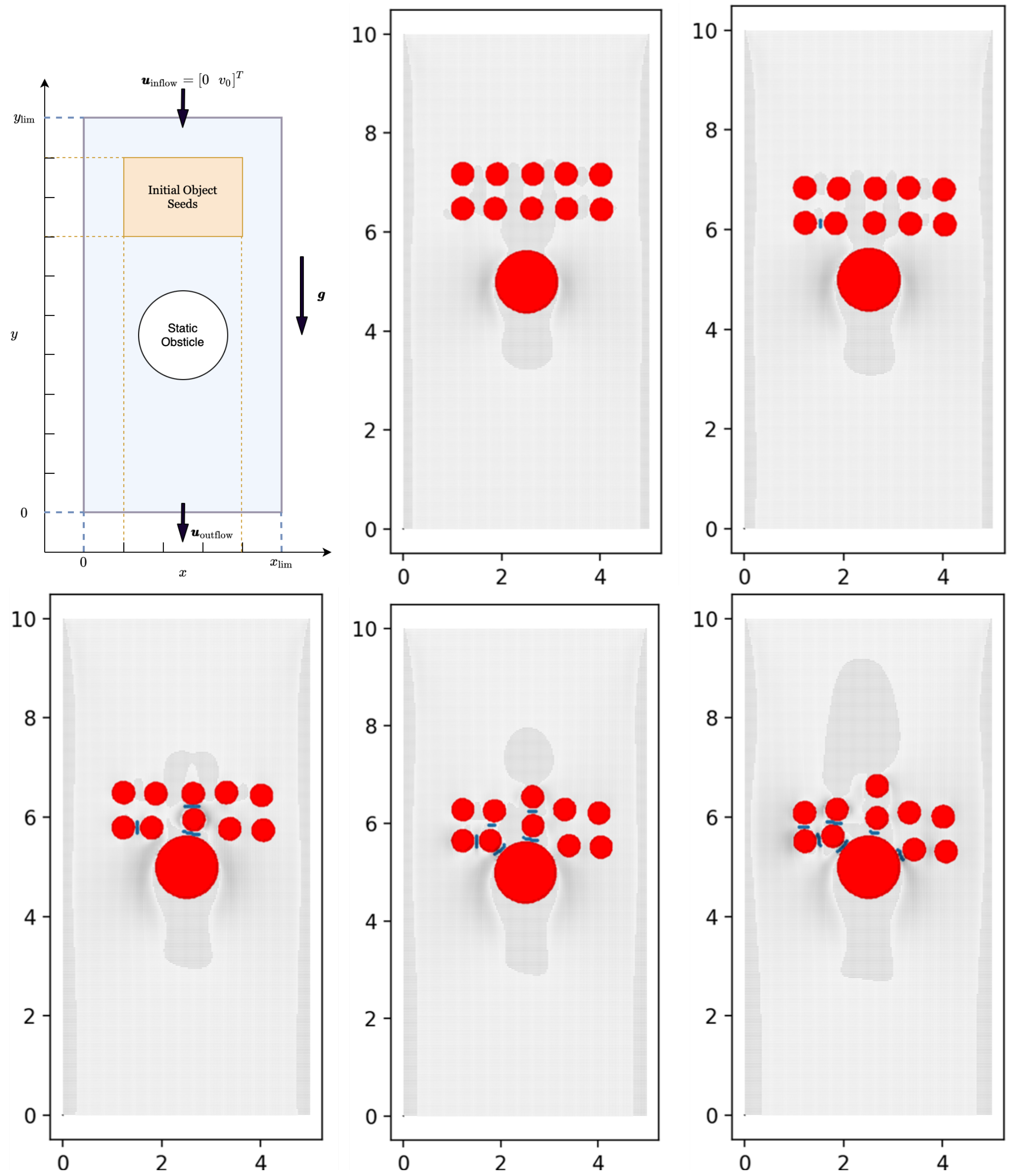

Figure 6.3: Tumbling Scenario for 10 falling circular objects over time $\boldsymbol{N} = (80, 160)$ running to $t = 10$.

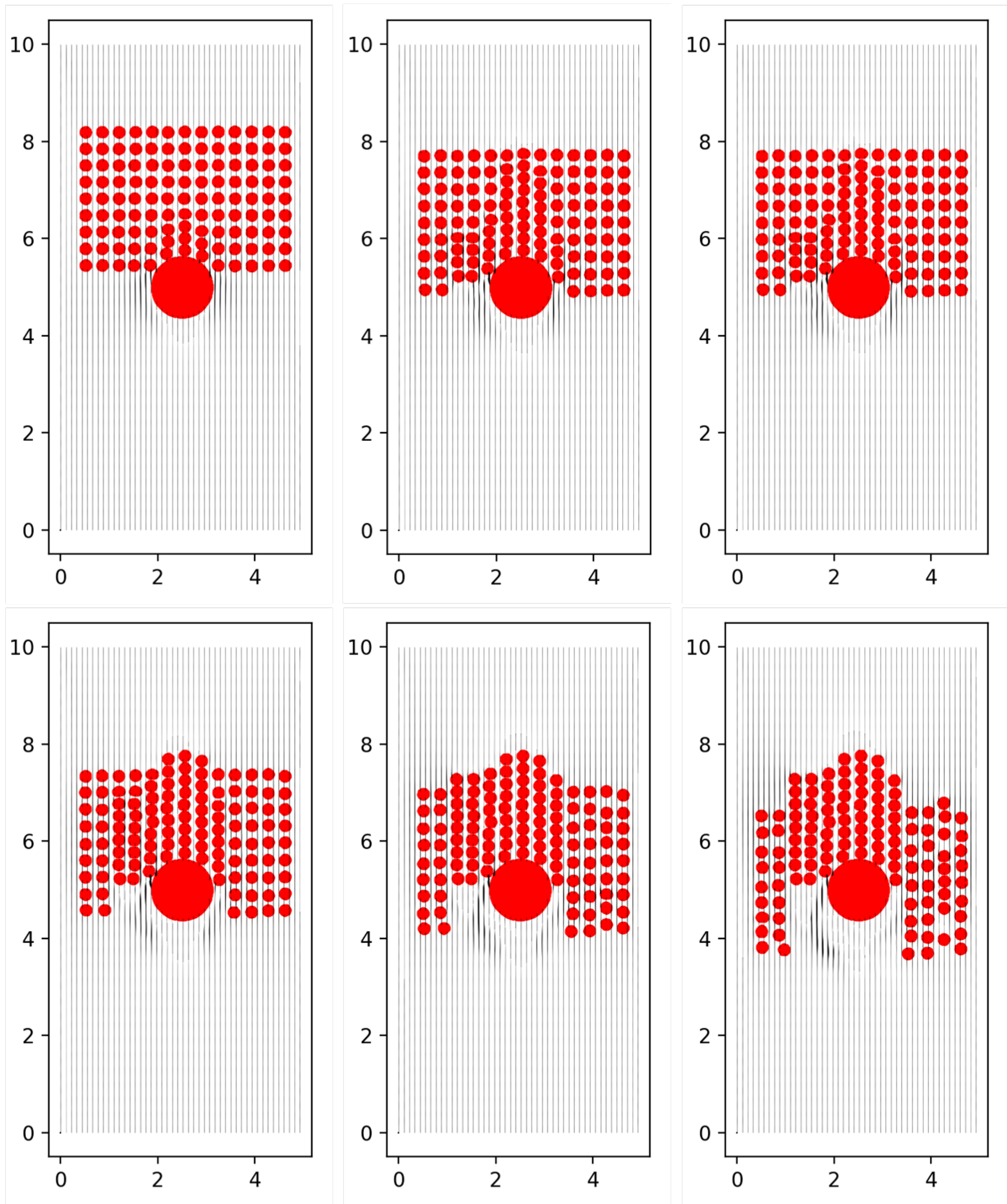

Figure 6.4: Scenario 2 with Dense (118) Falling Objects with grid size $\boldsymbol{N} = (320, 640)$ running to $t = 10$.

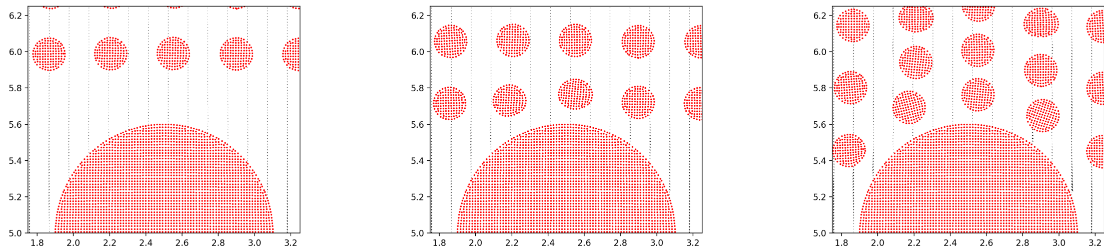

Figure 6.5: Close-up of initial collision point in Figure 6.4 at times $t = 0.5, 1, 1.5$ (from left-to-right).

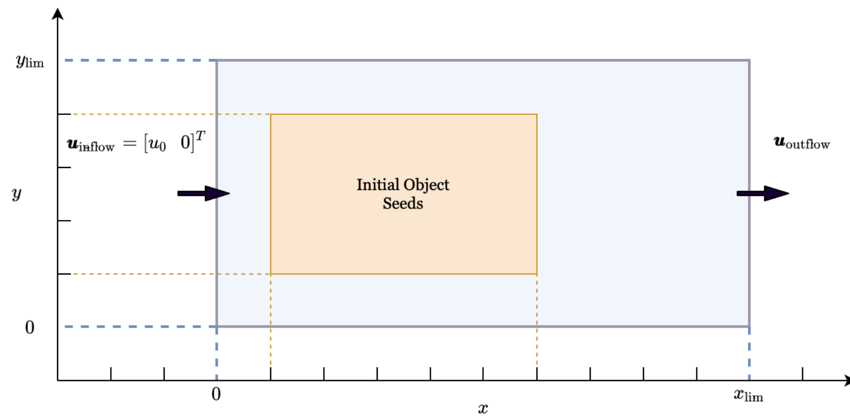

Figure 6.6: Scenario 3 diagram.

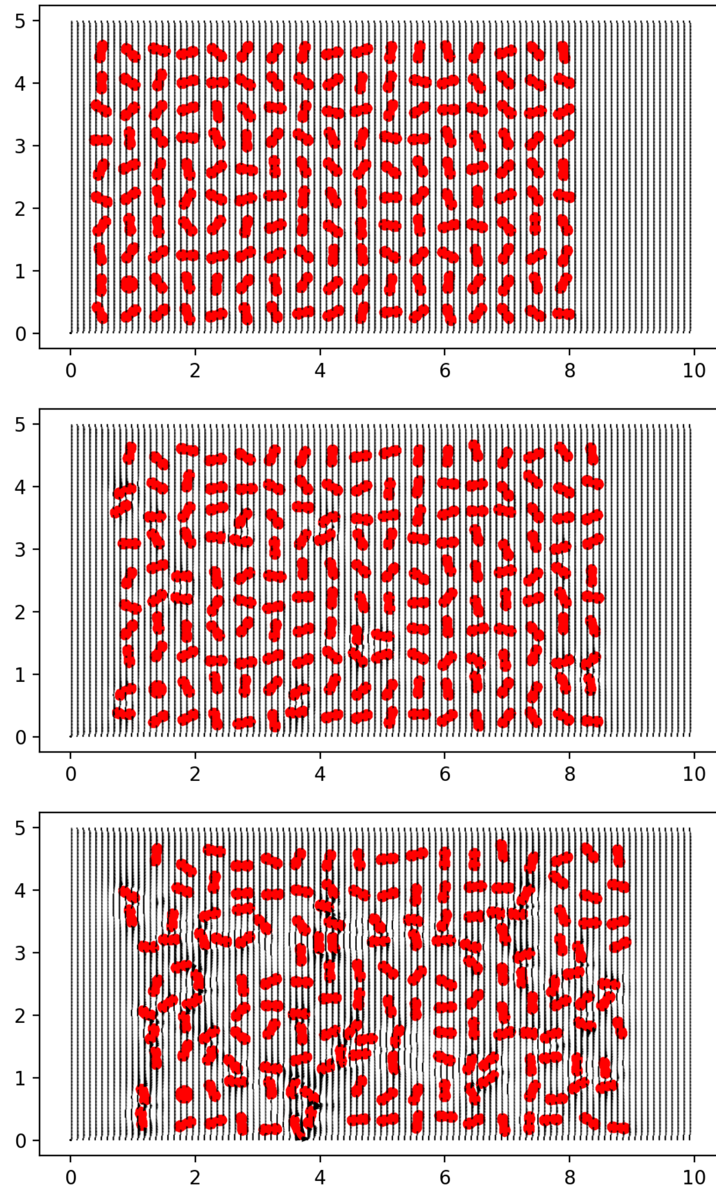

Figure 6.7: Scenario 3 realization. $\boldsymbol{N} = (320, 160)$ with 40 immersed objects. Pictured at instances $t = 3.35, 6.65, 10$ (top to bottom).

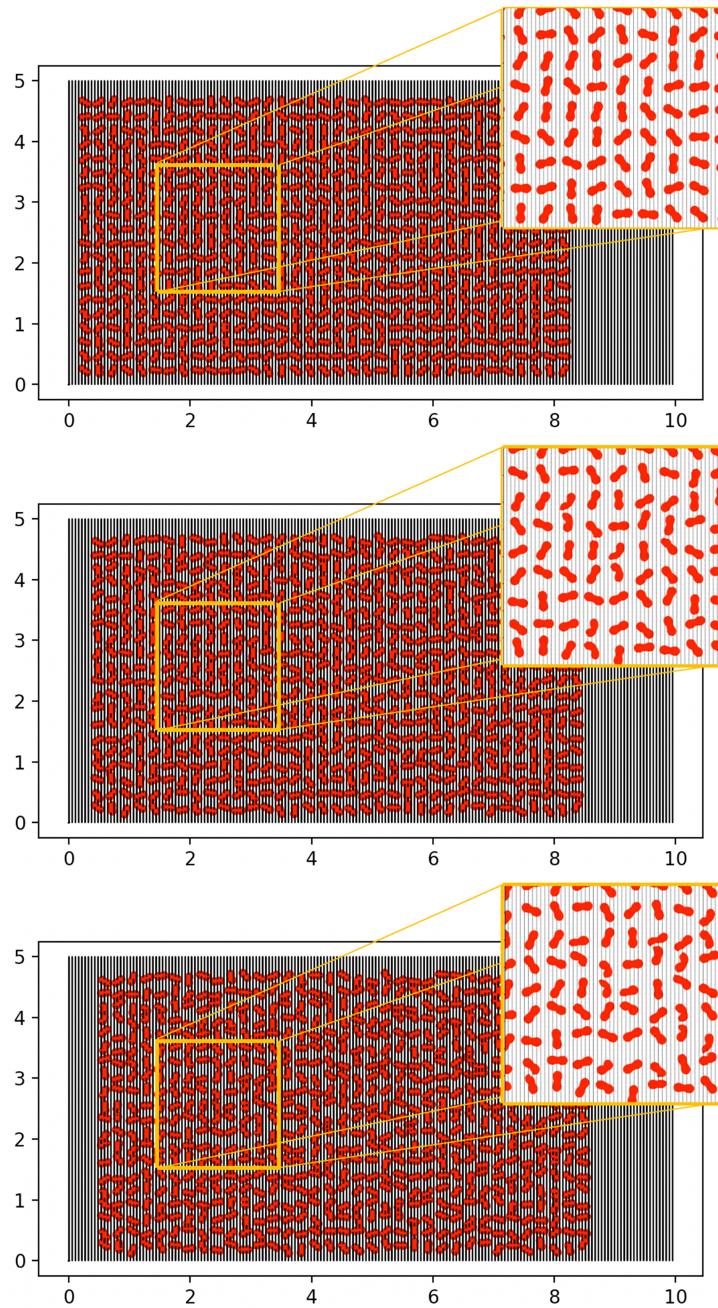

Figure 6.8: Scenario 3 realization. $\boldsymbol{N} = (320, 160)$ with 700 immersed objects. Pictured at instances $t = 3.35, 6.65, 10$ (top to bottom).

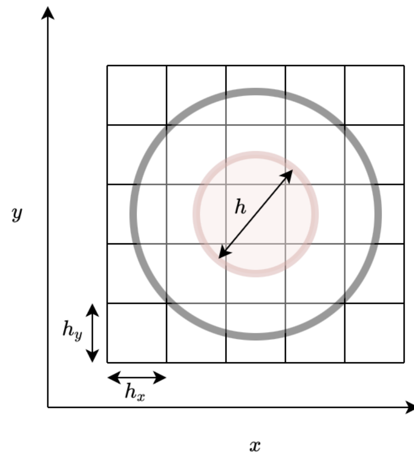

Figure 6.9: Visualization of the minimum object size for a simulation with grid cell size h.

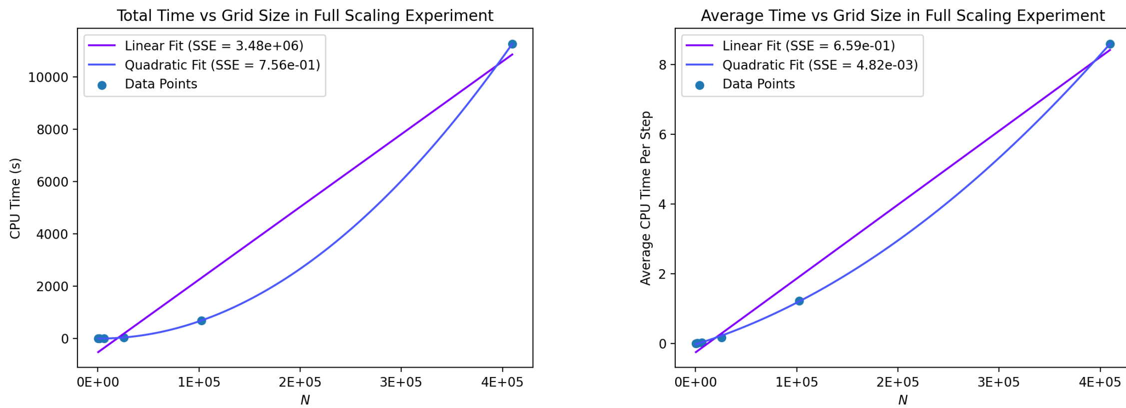

Figure 6.10: Scenario 1 - Plot of the total (left) and average per-step (right) CPU time to reach t_{end}.

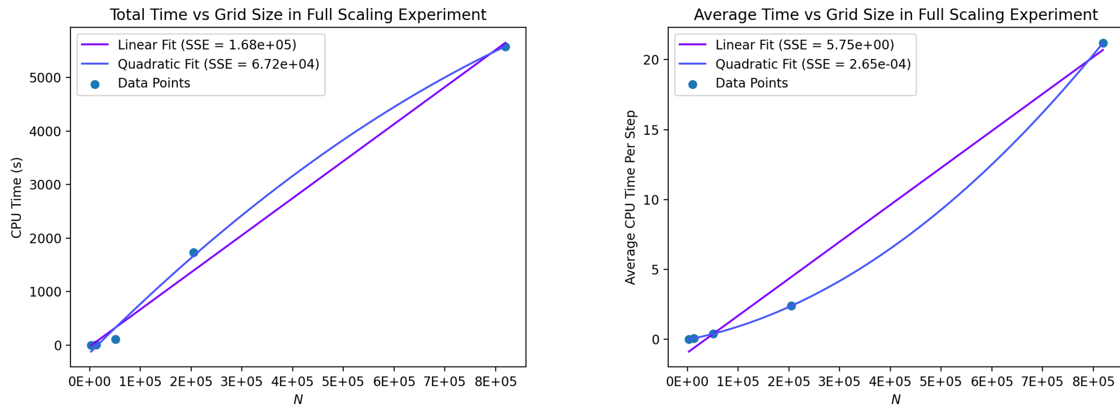

Figure 6.11: Scenario 2 - Plot of the total (left) and average per-step (right) CPU time to reach t_{end}.

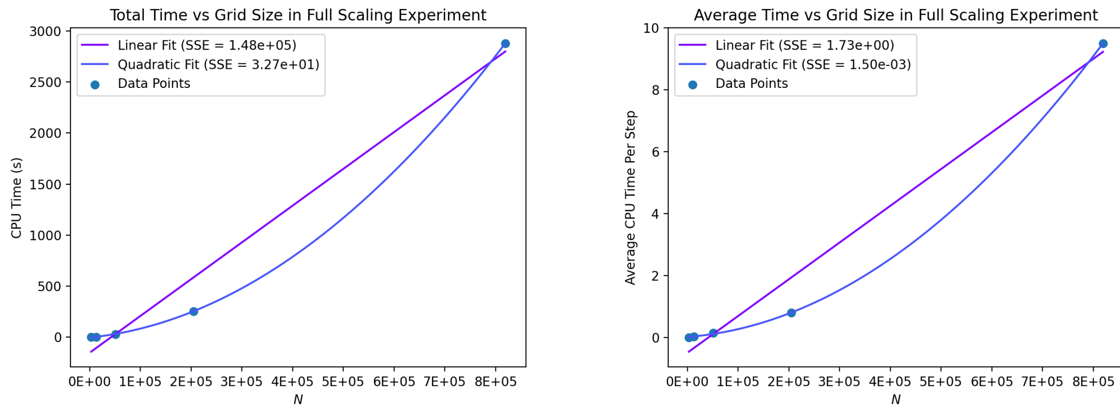

Figure 6.12: Scenario 3 - Plot of the total (left) and average per-step (right) CPU time to reach t_{end}.

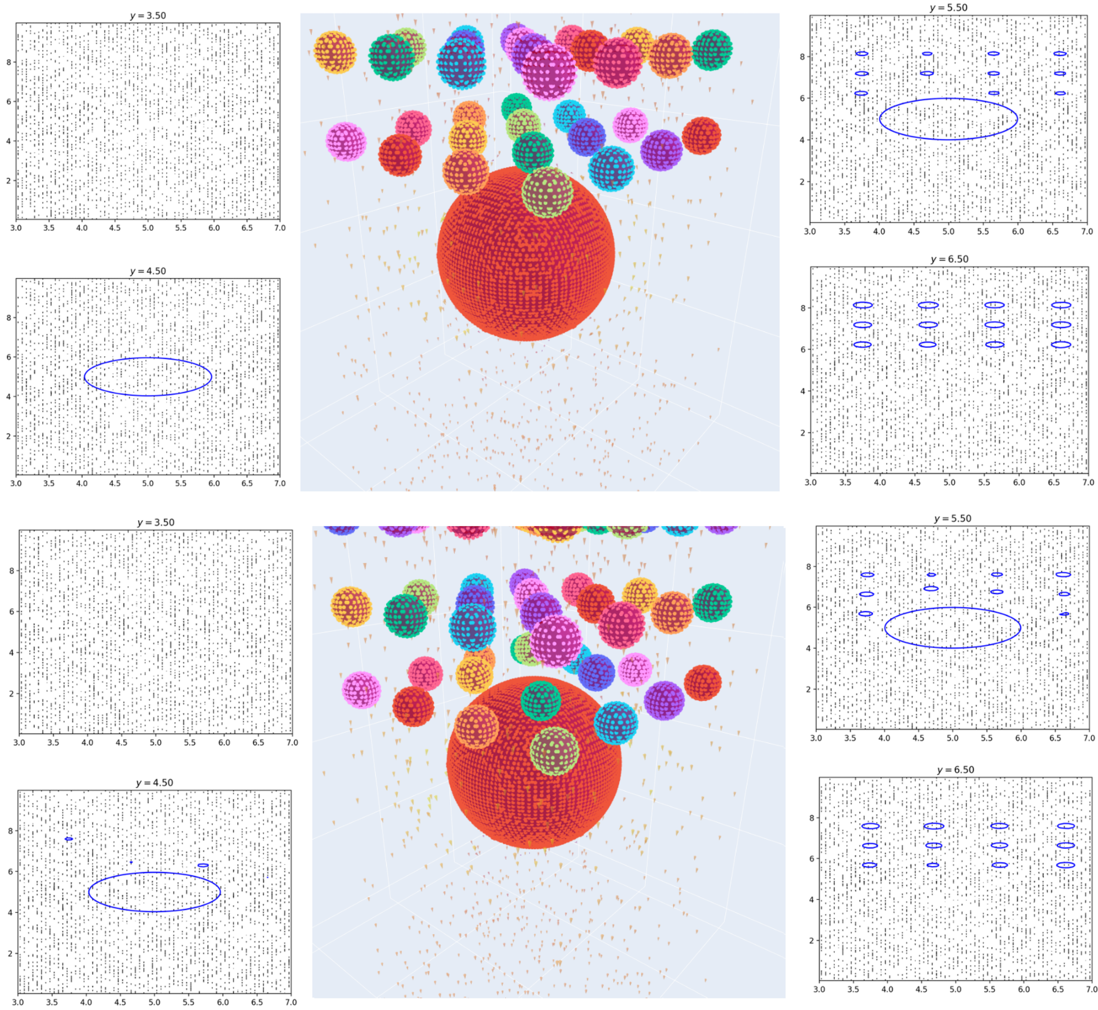

Figure 6.13: Scenario 2 Basic Example with a grid of size $\boldsymbol{N} = (160, 160, 160)$ and 49 immersed objects at $t = 5$ (top) and $t = 10$ (bottom)

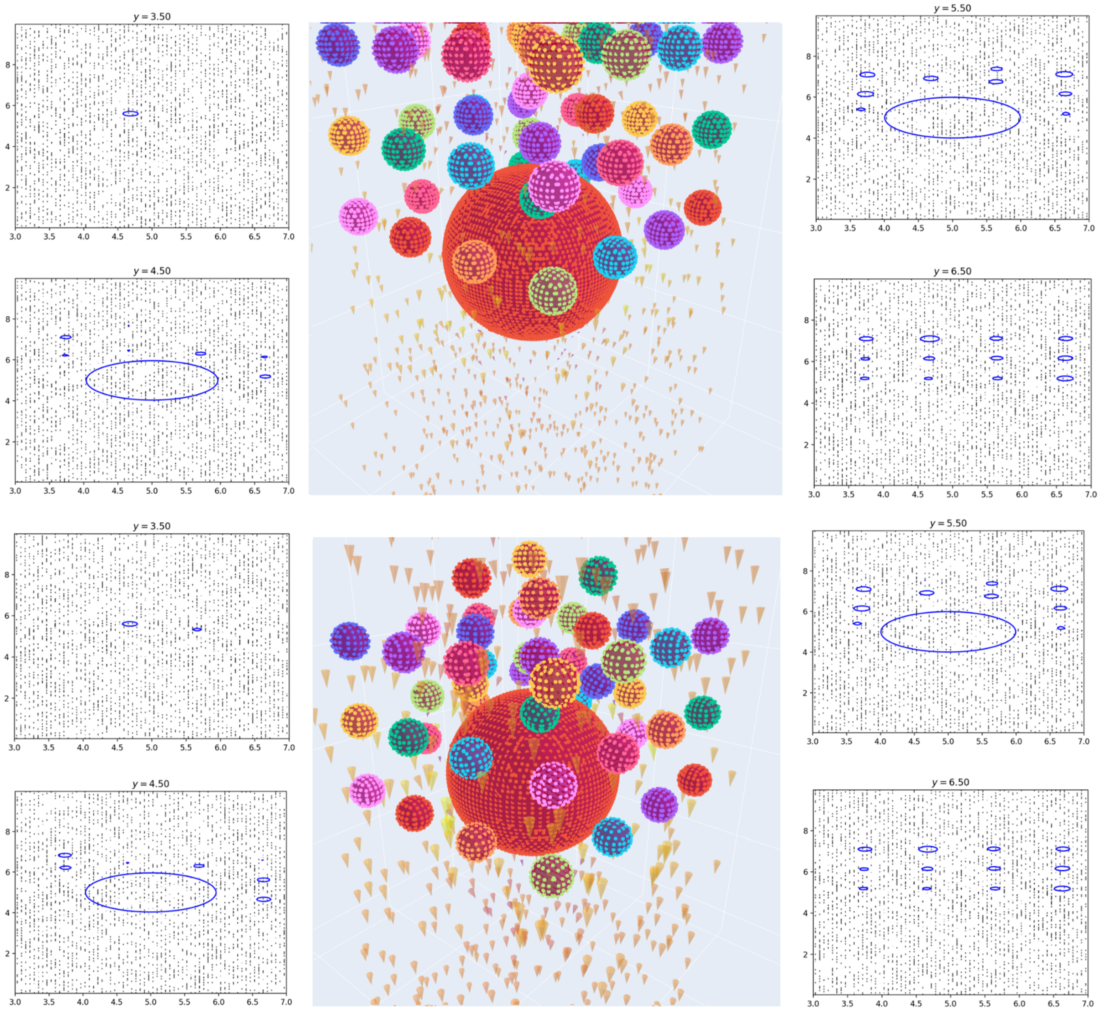

Figure 6.14: Scenario 2 Basic Example with Course Grid and 49 immersed objects at $t = 15$ (top) and $t = 20$ (bottom)

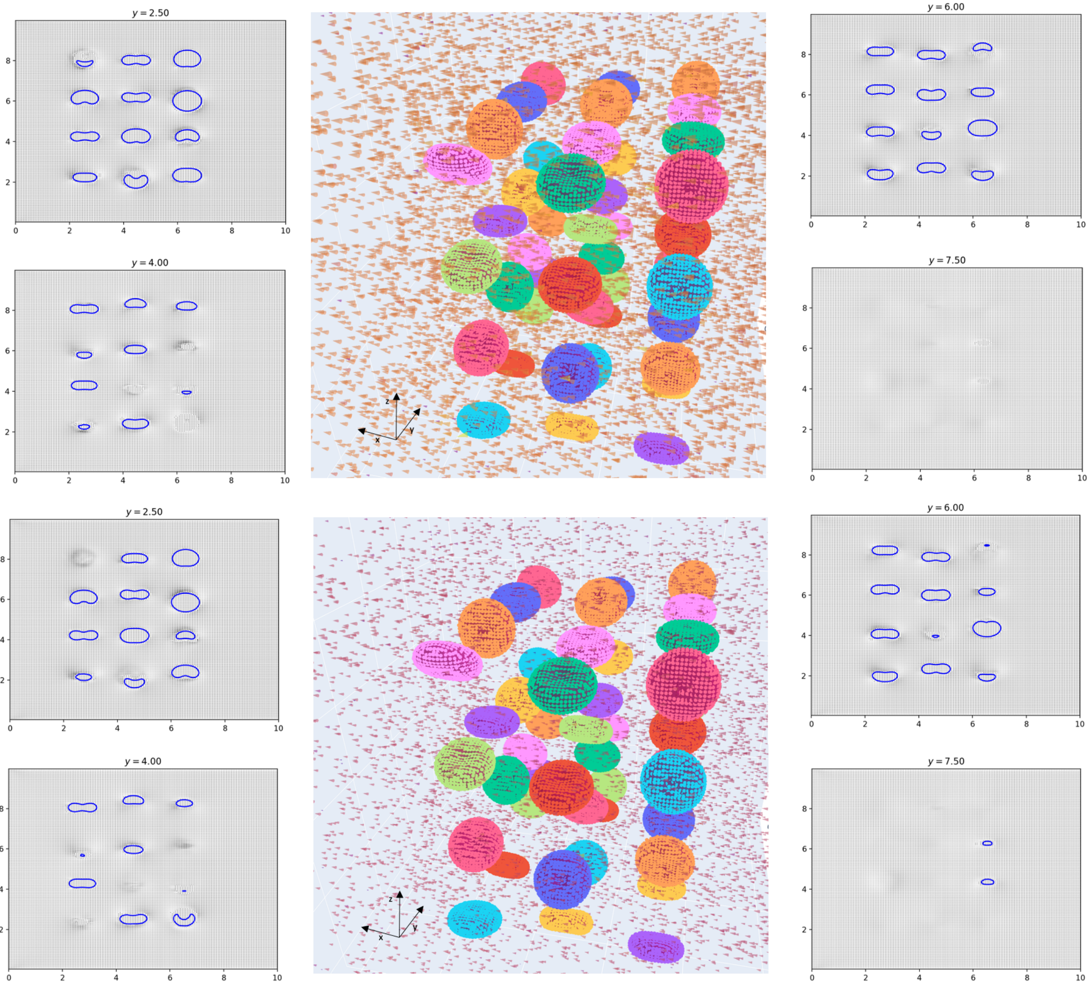

Figure 6.15: Scenario 3 Basic Example with Course Grid and 49 immersed objects at $t = 2.5$ (top) and $t = 5$ (bottom)

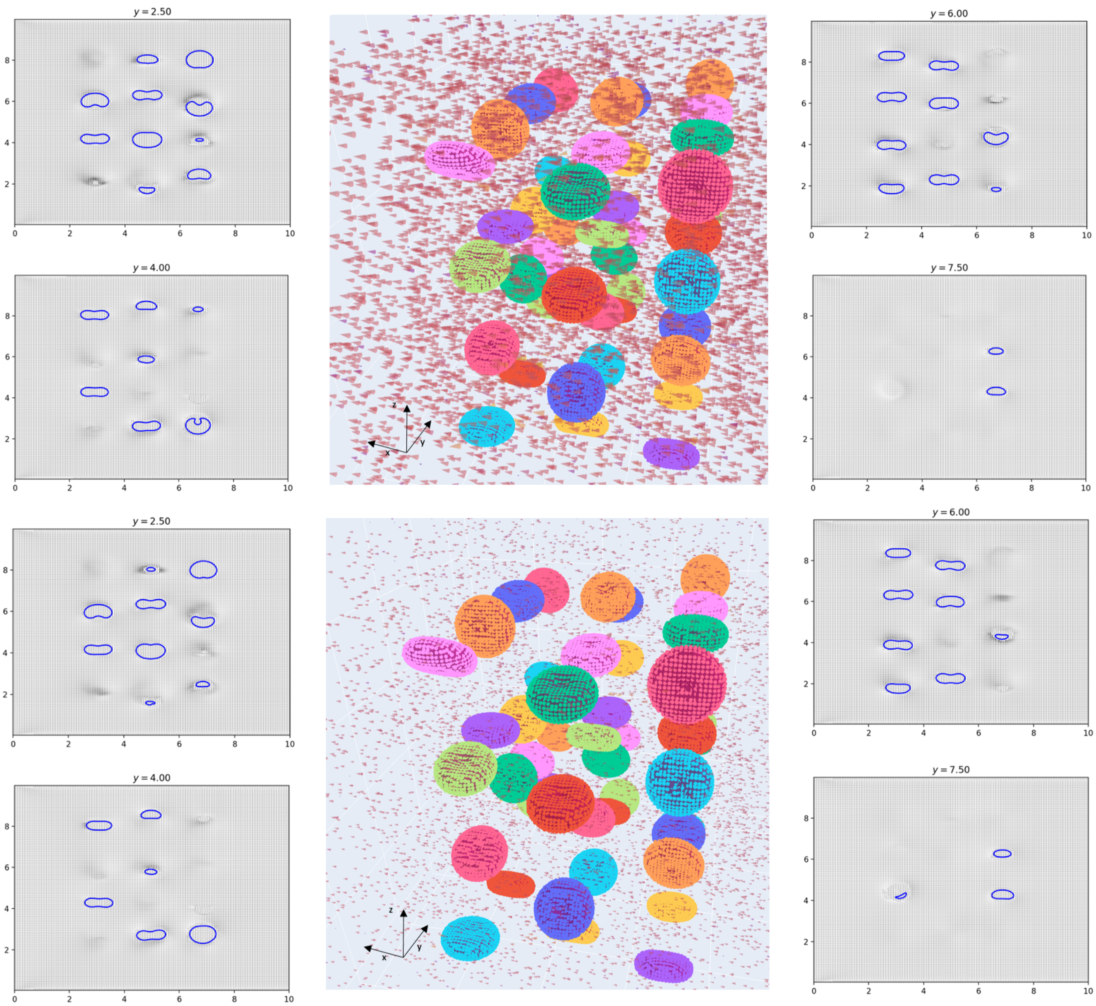

Figure 6.16: Scenario 3 Basic Example with Course Grid and 49 immersed objects at $t = 7.5$ (top) and $t = 10$ (bottom) (Give more details on each of the scenarios in-text)

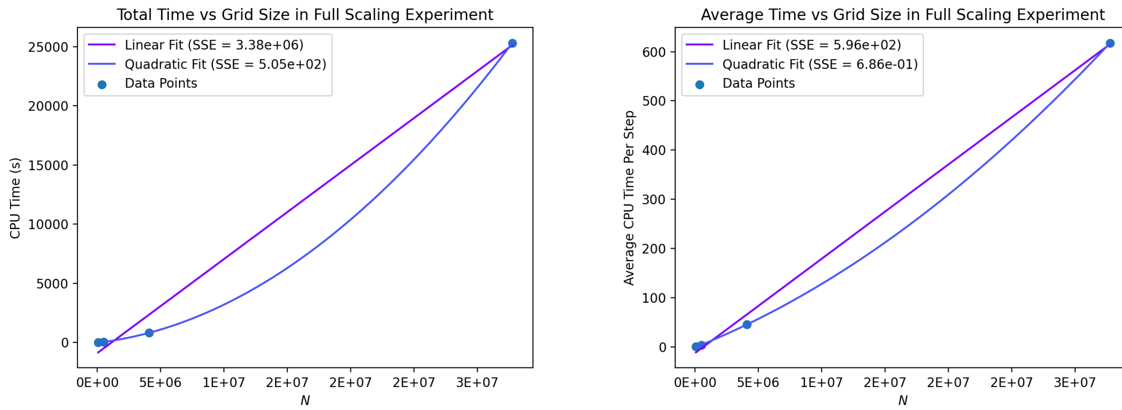

Figure 6.17: Simultaneous scaling results for Scenario Two in 3D.

Chapter 7

Conclusions and Future Work

In this thesis, we have presented a novel method for solving FSI problems including a large number of deformable objects in an incompressible fluid. We have derived the method from standard techniques in CFD, combined with novel heuristics to accommodate the specific difficulties of this general problem class. We presented a number of results, applying our method to a variety of complex scenarios in two and three dimensions to demonstrate the efficacy of our method. Scaling performance was evaluated and we found that the cost of the algorithm was largely independent of the number of objects that were immersed in the fluid. We hypothesized the cause of this, with our conclusions pointing to the fact that the Poisson equation for the fluid pressure becomes easier the more objects are placed in the fluid.

There are many avenues for future work. The most obvious is the application of this method to the problem it was designed to handle, namely the multi-scale simulation of blood flow using a realistic density of immersed red blood cells. Additionally, to further improve the performance GPU acceleration of the component algorithms is an obvious direction. This has begun common direction for numerical methods for fluids, with fast parallel methods begin developed for the solution of the Poisson equation. Our profiling results show this would provide by far the greatest performance increase for our method. Additionally, the exploration of alternative algorithms for the fluid component of the algorithm, such as those used in [24] [52] could be explored, which avoid the typical pressure solve through the use of heuristics.

References

[1] D Adalsteinsson and J. A Sethian. The Fast Construction of Extension Velocities in Level Set Methods. *Journal of Computational Physics*, 148(1):2–22, January 1999.

[2] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics*, 26(3):100, July 2007.

[3] Muhammad Mubashir Bhatti, Marin I Marin, Ahmed Zeeshan, and Sara I Abdelsalam. recent trends in computational fluid dynamics. *Frontiers in Physics*, 8:453, 2020.

[4] Ernest William Billington and A Tate. The physics of deformation and flow. *McGraw-Hill Book Co.*, 1981.

[5] Jose Luis Blanco. nanoflann.

[6] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM transactions on graphics (TOG)*, 33(4):1–11, 2014.

[7] Robert Bridson. *Fluid Simulation for Computer Graphics*. A K Peters/CRC Press, November 2018.

[8] David L Chopp. Some improvements of the fast marching method. *SIAM Journal on Scientific Computing*, 23(1):230–244, 2001.

[9] Alexandre Joel Chorin. Numerical Solution of the Navier-Stokes Equations. *Mathematics of Computation*, 22(104):745–762, 1968. Publisher: American Mathematical Society.

[10] Jean Donea, S Giuliani, and Jean-Pierre Halleux. An arbitrary lagrangian-eulerian finite element method for transient dynamic fluid-structure interactions. *Computer methods in applied mechanics and engineering*, 33(1-3):689–723, 1982.

[11] William Nichols Findley, James S Lai, Kasif Onaran, and RM Christensen. *Creep and relaxation of nonlinear viscoelastic materials with an introduction to linear viscoelasticity*. North Holland, 1977.

[12] Susan Fisher and Ming C Lin. Deformed distance fields for simulation of non-penetrating flexible bodies. In *Computer Animation and Simulation 2001*, pages 99–111. Springer, 2001.

[13] Arnulph Fuhrmann, Gerrit Sobotka, and Clemens Groß. Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon*, volume 2003, pages 58–65, 2003.

[14] Yuan-cheng Fung. Foundations of solid mechanics. *ENGLEWOOD CLIFFS, N. J., PRENTICE-HALL, INC., 1965. 525 P*, 1965.

[15] Olivier Génevaux, Arash Habibi, and Jean-Michel Dischler. Simulating fluid-solid interaction. In *Graphics Interface*, volume 2003, pages 31–38. Citeseer, 2003.

[16] Frederic Gibou, Ronald Fedkiw, and Stanley Osher. A review of level-set methods and some recent applications. *Journal of Computational Physics*, 353:82–109, 2018.

[17] Sigal Gottlieb and Chi-Wang Shu. Total variation diminishing runge-kutta schemes. *Mathematics of computation*, 67(221):73–85, 1998.

[18] Michael Griebel, Thomas Dornseifer, and Tilman Neunhoeffer. *Numerical Simulation in Fluid Dynamics*. Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics, January 1998.

[19] Jean-Luc Guermond and Peter Minev. High-order time stepping for the incompressible navier–stokes equations. *SIAM Journal on Scientific Computing*, 37(6):A2656–A2681, 2015.

[20] 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.

[21] Ping He and Rui Qiao. A full-Eulerian fluid–structure interactions. *Microfluidics and Nanofluidics*, 11(5):557–567, November 2011.

[22] Wolfgang Heidrich. Computing the barycentric coordinates of a projected point. *Journal of Graphics Tools*, 10(3):9–12, 2005.

[23] Gene Hou, Jin Wang, and Anita Layton. Numerical Methods for Fluid-Structure Interaction — A Review. *Communications in Computational Physics*, 12(2):337–377, August 2012.

[24] Christos Kotsalos, Jonas Latt, and Bastien Chopard. Bridging the computational gap between mesoscopic and continuum modeling of red blood cells for fully resolved blood flow. *Journal of Computational Physics*, 398:108905, December 2019. arXiv: 1903.06479.

[25] Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen. The lattice boltzmann method. *Springer International Publishing*, 10(978-3):4–15, 2017.

[26] GK Landau and EM Lifshitz. *Fluid Mechanics*. Elsevier, 1987.

[27] Alexander Leichner, Heiko Andrä, and Bernd Simeon. A contact algorithm for voxel-based meshes using an implicit boundary representation. *Computer Methods in Applied Mechanics and Engineering*, 352:276–299, 2019.

[28] Alexander Leichner, Heiko Andrä, and Bernd Simeon. A contact algorithm for voxel-based meshes using an implicit boundary representation. *Computer Methods in Applied Mechanics and Engineering*, 352:276–299, August 2019.

[29] Randall J LeVeque et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[30] David IW Levin. Csc417 lecture notes - university of toronto.

[31] Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):1–7, 2013.

[32] Kjetil O Lye, Siddhartha Mishra, and Deep Ray. Deep learning observables in computational fluid dynamics. *Journal of Computational Physics*, 410:109339, 2020.

[33] W Malalasekera and HK Versteeg. *An introduction to computational fluid dynamics*. Prentice Hall, 2007.

[34] Joe J Monaghan. Smoothed particle hydrodynamics. *Reports on progress in physics*, 68(8):1703, 2005.

[35] Takayuki Nagata, Mamoru Hosaka, Shun Takahashi, Ken Shimizu, Kota Fukuda, and Shigeru Obayashi. A simple collision algorithm for arbitrarily shaped objects in particle-resolved flow simulation using an immersed boundary method. *International Journal for Numerical Methods in Fluids*, 92(10):1256–1273, 2020.

[36] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153. Springer Science & Business Media, 2006.

[37] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[38] M. Overby, G. E. Brown, J. Li, and R. Narain. ADMM $\supseteq$ Projective Dynamics: Fast Simulation of Hyperelastic Models with Dynamic Constraints. *IEEE Transactions on Visualization and Computer Graphics*, 23(10):2222–2234, October 2017. Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[39] Charles S Peskin. Flow patterns around heart valves: a numerical method. *Journal of computational physics*, 10(2):252–271, 1972.

[40] Charles S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, January 2002. Publisher: Cambridge University Press.

[41] Thomas Richter. *Fluid-structure interactions: models, analysis and finite elements*, volume 118. Springer, 2017.

[42] Giovanni Russo and Peter Smereka. A remark on computing distance functions. *Journal of computational physics*, 163(1):51–67, 2000.

[43] James A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[44] James Albert Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge university press, 1999.

[45] Mark Sussman, Peter Smereka, and Stanley Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational physics*, 114(1):146–159, 1994.

[46] A. Telea and J. V. Wijk. An Augmented Fast Marching Method for Computing Skeletons and Centerlines. In *VisSym*, 2002.

[47] Murilo F. Tome and Sean McKee. GENSMAC: A Computational Marker and Cell Method for Free Surface Flows in General Domains. *Journal of Computational Physics*, 110(1):171–186, January 1994.

[48] Luminita A Vese and Tony F Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *International journal of computer vision*, 50(3):271–293, 2002.

[49] Andrew Witkin and David Baraff. Physically based modeling. *SIGGRAPH '97 Course Notes*.

[50] Lang Xu, Yuhua Lu, and Qian Liu. Integrating viscoelastic mass spring dampers into position-based dynamics to simulate soft tissue deformation in real time. *Royal Society open science*, 5(2):171587, 2018.

[51] Myoungho Yoon, Gangjoon Yoon, and Chohong Min. On solving the singular system arisen from poisson equation with neumann boundary condition. *Journal of Scientific Computing*, 69(1):391–405, 2016.

[52] Gábor Závodszky, Britt van Rooij, Victor Azizi, Saad Alowayyed, and Alfons Hoekstra. Hemocell: a high-performance microscopic cellular library. *Procedia Computer Science*, 108:159–165, December 2017.