Parallel contact-aware algorithms for large-scale direct blood flow simulations

by

Libin Lu

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

May 2019

Denis Zorin

Dedication

To my wife, parents, grandparents and daughter.

Acknowledgements

I would like to thank my academic advisor, Denis Zorin, for introducing me to the world of scientific computing, for his continuous guidance, patience, and support throughout my graduate study. I also thank Abtin Rahimian, for all his help through the research projects, for being always available. I am grateful to Matthew Morse, for a fruitful collaboration, I could not have done any of this on my own. I also would like to thank Shravan Veerapareni, for his mentorship and valuable discussions in the last semester. I would like to thank Georg Stadler, for his help on Chapter 4 and serving in my thesis committee. I would like to thank Marsha Berger, for her support on the work of Chapter 2 and serving in my thesis committee. I am also thankful to Michael Overton for his support and valuable conversations about various aspects of Chapter 2. I also would like to thank Michael Shelley for serving in my thesis committee.

Abstract

Experimental and theoretical evidence suggests that blood flow can be well approximated by a model of a Newtonian fluid and deformable particles representing the red blood cells. We use a well-established boundary integral formulation for the problem as the foundation of our approach. This type of formulations, with a high-order spatial discretization and an implicit and adaptive time discretization, have been shown to be able to handle complex interactions between particles with high accuracy. Yet, for dense suspensions, very small time-steps or expensive implicit solves as well as a large number of discretization points are required to avoid non-physical contact and intersections between particles, leading to infinite forces and numerical instability. Given the importance of vesicle flows, in this thesis we focus in efficient numerical methods for such problems: we present computationally parallel-scalable algorithms for the simulation of dense deformable vesicles in two and three dimensions both in unbounded and bounded domain.

Our method maintains the accuracy of previous methods at a significantly lower cost for dense suspensions and the time step size is independent from the volume fraction. The key idea is to ensure interference-free configuration by introducing explicit contact constraints into the system. While such constraints are unnecessary in the formulation, in the discrete form of the problem, they make it possible to eliminate catastrophic loss of accuracy by preventing contact explicitly.

Introducing contact constraints results in a significant increase in stable time-step size for locally-implicit time-stepping, and a reduction in the number of points adequate for stability. Our method permits simulations with high volume fractions; we report results with up to 60% volume fraction. We demonstrated the parallel scaling of the algorithms on up to 35K CPU cores.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ability to simulate particulate flows faithfully with deformable and rigid particles suspended in viscous incompressible fluids has the potential to provide insight into complicated physiological processes. The most important example of which is blood flow simulation. Accurate blood flow simulation can be used to study biological physics such as blood flow clogging and blood cells separation. We model the blood flow as a particulate suspension of elastic membranes that resist bending and tension and are filled with a Newtonian fluid. The type of particles are generally known as vesicles. We use the terms vesicle and RBC interchangeably referring to this elastic capsule representation in the rest of this thesis. Vesicles are used to understand the properties of biomembranes [28, 63], and to simulate the motion of blood cells, in which vesicles with moderate viscosity contrast are used to model red blood cells and high viscosity contrast vesicles or rigid particles are used to model white blood cells [9].

However, direct simulation of blood flow is an extremely challenging task. Even simulating the blood flow in smaller vessels requires modeling millions of cells (one microliter of blood contains around five million RBCs) along with a complex blood vessel. RBCs are highly deformable and cannot be well-approximated by rigid particles. The volume fraction of cells in human blood flow reaches 45%, which means that a very large fraction of cells are in close contact with other cells or vessel walls at any given time. These constraints preclude a large number of discretization points per cell and make an evolving mesh of the fluid domain impractical and costly at large scale.

Simulations capable of capturing these various types of flows faithfully must be

- *accurate*, to reproduce the physics of interest without concern for numerical error;
- *robust*, to handle high-volume-fraction flows, close contact between cells and vessel walls, complex geometries, and long simulation times;
- *efficient and scalable*, to support a realistic number of cells in flows through complex blood vessels.

Nevertheless, simulating realistic human blood flow requires simulating *dense* suspensions of rigid and deformable particles. This entails many numerical challenges, one of which is the need to frequently and accurately resolve contact between particles, requiring very small times steps and/or fine spatial discretization. To make such simulations at larger scale practical, we present an efficient, accurate, and robust method for simulation of dense suspensions in Stokesian fluid in 2D and 3D both in bounded and unbounded domains. We use the boundary integral formulation to represent the flow and impose the contact-free condition as a constraint. This thesis focuses mainly on suspensions of rigid bodies and vesicles with high volume fractions, in which multiple particles are in contact or near-contact.

1.1 Our contributions

1. We present a contact-aware algorithm for particulate Stokesian suspensions simulation in 2D. For high volume fraction, our method makes it possible to increase the step size by at least an order of magnitude, and the simulation remain stable even for relatively coarse spatial discretizations.

2. We present a parallel contact-aware algorithm for particulate Stokesian suspensions simulation in 3D unbounded domain. Our method permits simulations with high volume fractions (we report results with up to 60% volume fraction). Within our framework, the time step size is independent from the volume fraction and the simulation wall-clock-time is at least an order of magnitude faster than the adaptive case.

3. We present a parallel platform for long-time simulations of red blood cells through complex blood vessels. Flows through several complicated geometries are demonstrated. We have parallelized a boundary solver for elliptic PDE's on smooth complex geometries in 3D. The sequential version of the boundary solver is an ongoing work of Matthew Morse, Abtin Rahimian and Denis Zorin. We extend the parallel collision handling to include arbitrary boundaries composed of patches. We present weak and strong scalability results of our simulation on the Stampede2 cluster at the Texas Advanced Computing Center along with several visualizations long-time, large-scale blood cell flows through vessels.

1.2 Outline of The Thesis

In **Chapter 2** of this thesis, we present an efficient, accurate, and robust method for simulation of dense suspensions of deformable and rigid particles immersed in Stokesian fluid in two dimensions. This chapter is the joint work with Abtin Rahimian and Denis Zorin[53]. We use a well-established boundary integral formulation for the problem as the foundation of our approach. This type of formulation, with a high-order spatial discretization and an implicit and adaptive time discretization, have been shown to be able to handle complex interactions between particles with high accuracy. For dense suspensions, very small time-steps or expensive implicit solves as well as a large number of discretization points are required to avoid non-physical contact and intersections between particles, leading to infinite forces and numerical instability. Our method maintains the accuracy of previous methods at a significantly lower cost for dense suspensions. The key idea is to ensure interference-free configuration by introducing explicit contact constraints into the system. While such constraints are unnecessary in the formulation, in the discrete form of the problem, they make it possible to eliminate catastrophic loss of accuracy by preventing contact explicitly. Introducing contact constraints results in a significant increase in stable time-step size for explicit time-stepping, and a reduction in the number of points adequate for stability.

In **Chapter 3** of this thesis, we present a parallel-scalable method for simulating non-dilute suspensions of deformable particles immersed in Stokesian fluid in three dimensions. This chapter is the joint work with Abtin Rahimian and Denis Zorin[54]. A critical component in these simulations is robust and accurate collision handling. This chapter complements Chapter 2 by extending it to 3D and by introducing new parallel algorithms for collision detection and handling.

Our method maintains the accuracy of previous methods at a significantly lower cost for dense suspensions and the time step size is independent from the volume fraction. Our method permits simulations with high volume fractions; we report results with up to 60% volume fraction. We demonstrated the parallel scaling of the algorithms on up to 16K CPU cores.

In **Chapter 4** of this thesis, we present a fast scalable platform for the simulation of red blood cell (RBC) flows through complex capillaries by modeling the physical system as a viscous fluid with immersed deformable particles. This chapter is the joint work with Matthew Morse, Abtin Rahimian, Georg Stadler and Denis Zorin[55]. We describe a parallel boundary integral equation solver for general elliptic partial differential equations, which we apply to Stokes flow through blood vessels. We also detail a parallel collision avoiding algorithm to ensure RBCs and the blood vessel remain contact-free. We have scaled our code on Stampede2 at the Texas Advanced Computing Center up to 34,816 cores. Our largest simulation enforces a contact-free state between four billion surface elements and solves for three billion degrees of freedom on one million RBCs and a blood vessel composed from two million patches.

Chapter 2

Contact-aware simulations of particulate Stokesian suspensions in two-dimensional domain

2.1 Introduction

Particulate Stokesian suspensions of deformable and rigid particles commonly occur in nature and are widely used in industrial applications. Examples of such fluids include emulsions, colloidal structures, particulate suspensions, and blood. Most of these examples are *complex fluids*, i.e., fluids with unusual macroscopic behavior, often defying a simple constitutive-law description. A major challenge in understanding the physics of complex fluids is the link between microscopic and macroscopic fluid behavior. Dynamic simulation is a powerful tool [10, 64] to gain insight into the underlying physical principles that govern these suspensions and to obtain relevant constitutive relationships.

Nevertheless, simulating *dense* suspensions of rigid and deformable particles entails many numerical challenges, one of which is the need to frequently and accurately resolve contact between particles, requiring very small times steps and/or fine spatial discretization. To make such simulations at larger scale practical, we present an efficient, accurate, and robust method for simulation of dense suspensions in Stokesian fluid in 2D (e.g., Fig. 2.7), which does not make any assumption about the dimensions of the problem and is extendable to 3D. We use the boundary integral formulation to represent the flow and impose the contact-free condition as a constraint. This work focuses mainly on suspensions of rigid bodies and vesicles with high volume fractions, in which multiple particles are in contact or near-contact.

Vesicles are closed deformable membranes suspended in a viscous medium. The dynamic deformation of vesicles and their interaction with the Stokesian fluid play an important role in many biological phenomena. They are used to understand the properties of biomembranes [28, 63], and to simulate the motion of blood cells, in which vesicles with moderate viscosity contrast are used to model red blood cells and high viscosity contrast vesicles or rigid particles are used to model white blood cells [9].

Boundary integral formulations offer a natural approach for accurate simulation of vesicle flows, by reducing the problem to solving equations on surfaces, and eliminating the need for discretizing changing 3D volumes. However, in non-dilute suspensions, these methods are hindered by difficulties: inaccuracies in computing near-singular integrals, and artificial force singularities caused by (non-physical) intersection of particles. Contact situations in the Stokesian particulate flows occur frequently when the volume fraction of suspensions is high, viscosity contrast of vesicles is high, or rigid particles are present. On the other hand, there are certain classes of flows and formulations that are not hindered by frequent particle collision, e.g., unbounded flow of vesicles with no viscosity contrast [103]. The dynamics of particle collision in Stokes flow are governed by the lubrication film formation and drainage, which has a time scale much shorter than that of the flow [27]. Solely relying on the hydrodynamics to prevent contact requires the accurate solution of the flow in the lubrication film, which in turn entails very fine spatial and temporal resolution accompanied by increasingly ill-conditioned linear systems in the boundary integral setting [85, 88] — imposing excessive computational burden as the volume fraction increases.

While adaptive time-stepping [82, 83] goes a long way in maintaining stability and efficiency in dilute suspensions, the time-step is determined by the closest pair of vesicles, and tends to be uniformly small for dense suspensions.

In this work we take a different approach: we augment the governing equations with the contact constraint. While from the point of view of the physics of the problem such a constraint is redundant, as non-penetration is ensured by fluid forces, in numerical context it plays an important role, improving both robustness and accuracy of simulations. Typically, a contact law/constraint is characterized by conditions of non-penetration, no-adhesion as well as a mechanical complementarity condition, i.e., the contact force is zero when there is no collision. These three conditions are known as Signorini conditions in the context of contact mechanics or KKT conditions in the context of constrained optimization [66, 110].

2.1.1 Our contributions

Contact constraints ensure that the discretized system remains intersection-free, even for relatively coarse spatial and temporal discretizations, where the fidelity of the numerical model is insufficient for resolving the lubrication film. These constraints lead to a Nonlinear Complementarity Problem (NCP), which we linearize and solve using an iterative method that avoids explicit construction of full matrices. We describe an implicit-explicit time-stepping scheme, adapting Spectral Deferred Correction (SDC) to our constrained setting, Section 2.3.2.

Contact constraints control the minimum distance between vesicles, maintaining it independent of the temporal resolution. While solving NCP at every step incurs an additional cost, it is more than compensated by the ability of our method to maintain larger time-steps, and lower spatial resolutions for a given target error.

For high volume fraction, our method makes it possible to increase the step size by at least an order of magnitude, and the simulation remain stable even for relatively coarse spatial discretizations (16 points per vesicle, versus at least 64 needed for stability without contact resolution; Section 2.4).

2.1.2 Synopsis of the method

We use the boundary integral formulation based on [82, 86, 103]; the basic formulation uses integral equation form of the problem and includes the effects of the viscosity contrast, fixed boundaries, as well as deformable and rigid moving bodies. We add contact constraints to this formulation, as an inequality constraint on a gap function that is based on *space-time intersection volume* [36]. The contact force is then parallel to the gradient of this volume with the Lagrange multiplier as its magnitude. We solve the contact NCP for the Lagrange multipliers of the constraints using a Newton-like matrix-free method, as a sequence of Linear Complementarity Problems (LCP) [15, 21], with each solved iteratively using GMRES. The spectral Fourier bases are used for spatial discretization. For time stepping, we use semi-implicit backward Euler or semi-implicit Spectral Deferred Correction (SDC).

2.1.3 Related work

Related work on Stokesian particle flows Stokesian particle models are employed to theoretically and experimentally investigate the properties of biological membranes [92], drug-carrying capsules [97], and blood cells [67, 76]. There is an extensive body of work on numerical methods for Stokesian particulate flows and an excellent review of the literature up to 2001 can be found in [78]. Reviews of later advances can be found in [86, 88, 103]. Here, we briefly summarize the most important numerical methods and discuss the most recent developments.

Integral equation methods have been used extensively for the simulation of Stokesian particulate flows such as droplets and bubbles [51, 52, 89, 123], vesicles [22, 25, 76, 86, 88, 94, 103, 120, 121], and rigid particles [74, 75, 119]. Other methods — such as phase-field approach [11, 18], immersed boundary and front tracking methods [43, 113], and level set method [47] — are used by several authors for the simulation of particulate flows.

For certain flow regimes, near interaction and collision of particles has been a source of difficulty, which was addressed either by spatial and temporal refinement to resolve the correct dynamics (increasing the computational burden) or by the introduction of repulsion forces (making the time-stepping stiff).

[93] presented a framework for dynamic simulation of rigid particles with spherical or cylindrical shapes, in which the lubrication forces were included directly by putting Stokes doublets at the contact midpoint. The magnitude of lubrication force was computed using asymptotic analysis. To maintain the accuracy in the interaction of deformable drops, [51, 124, 125] resorted to time-step refinement where the time step is kept proportional to particle distance d. [124, 125] keep the time step proportional to $\sqrt{d}$. [51] adjust both the grid spacing around the contact region and the time-step to be proportional to d. [25] resorted to repulsion force to avoid contact in a 2D particulate flow. In a later work for 3D, Freund and coauthors [122] observed that significantly larger repulsion force density are needed in three dimensions, as the total repulsion force is distributed over a smaller region, when measured as a fraction of the total surface area/length. Consequently, they used a purely kinematic collision handing, in which, after each time-step, the intersecting points are moved outside.

[83] applies adaptive time-stepping and backtracking to resolve collisions. Similarly, [68] present an interesting integral equation method for the flow of droplets in two dimensions with a specialized quadrature scheme for accurate near-singular evaluation enabling simulation of flows with close to touching particles. While methods using adaptivity both in space and time are the most robust and accurate, they incur excessive cost as means of collision handling.

Related work on contact response A broad range of methods were developed for collision detection and response. While the work in contact mechanics often focuses on capturing the physics of the contact correctly (e.g., taking into account friction effects), the work in computer graphics literature emphasizes robustness and efficiency. In our context, robustness and efficiency are particularly important, as we aim to model vesicle flows with high volume fraction and large number of particles. Physical correctness has a somewhat different meaning: as we know that if the forces and surfaces in the system are resolved with high accuracy, the contacts would not occur, our primary emphasis is on reducing the impact of the artificial forces associated with contacts on the system.

There is an extensive literature on contact handling in computational contact mechanics mainly in the context of FEM mechanical and thermal analysis [24, 39, 46, 81, 102, 109, 110]. [109, 110] presents in-depth reviews of the contact mechanics framework. The works in contact mechanics literature can be categorized based on their ability in handling large deformations and/or tangential friction. In some of the methods, to simplify the problem, small deformation assumption is used to predefine the active part of the boundary as well as to align the FEM mesh. Numerical methods for contact response can be categorized as (i) penalty forces, (ii) impulse/kinematic responses, and (iii) constraint solvers.

From algorithmic viewpoint, contact mechanics methods in FEM include: (i) Node-to-node methods where the contact between nodes is only considered. The FEM nodes of contacting bodies need to aligned and therefore this method is only applicable to small deformation. (ii) Node-to-surface methods check the collision between predefined set of nodes and segments. Similar to node-to-node methods, these methods can only handle small deformations. (iii) Surface-to-surface methods, where the contact constraint is imposed in weak form. In contrast to the two previous class of algorithms, methods in this class are capable of handling large deformations. Mortar Method is well-known within this class of algorithms [24, 46, 81, 102]. The Mortar Method was initially developed for connecting different nonmatching meshes in the domain decomposition approaches for parallel computing, e.g., [80].

In these methods, no-penetration is either enforced as a constraint using a Lagrangian (identified with the contact pressure) or penalty force based on a gap function. To the best of our knowledge, for contact mechanics problems, a signed distance between geometric primitives is used as the gap function, in contrast to our approach where we use space-time interference volume.

[24] present a frictionless contact resolution framework for 2D finite deformation using Mortar Method using penalty force or Lagrange multiplier. [102] use similar method for frictional contact in 2D. [81] use Mortar Method for large deformation contact using quadratic element.

Our problem has similarities to large-deformation frictionless contact problems in contact mechanics. An important difference however, is the presence of fluid, which plays a major role in contact response.

Application of boundary integral methods in contact mechanics is rather limited compared to the FEM methods [20, 33]. [20] used Boundary Element Method for the static contact problem where Coulomb friction is presented. [33] solved static problem with load increment and contact constraint on displacement and traction.

In computer graphics literature, a set of commonly used and efficient methods are based on [79], a method for the collision handling of mass-spring cloth models. To ensure that the system remains intersection-free, zones of impact are introduced and rigid body motion is enforced in each zone of impact; while this method works well in practice, its effects on the physics of the objects are difficult to quantify.

Penalty methods are common due to the ease in their implementation, but suffer from time-stepping stiffness and/or the lack of robustness. [8] uses implicit time-stepping coupled with repulsion force equal to the variation of the quadratic constraint energy with respect to control vertices. Soft collisions are handled by the introduction of damped spring and rigid collisions are enforced by modification to the mass matrix. [23] introduced Layered Depth Images to allow efficient computation of the collision volumes and their gradients using GPUs. A penalty force proportional to the gradient is used to resolve collisions. However, the stiffness of the repulsion force varies greatly (from 10^5 to 10^{10}) in their experiments. To address these difficulties, [35] present a framework for robust simulation of contact mechanics using penalty forces through asynchronous time-stepping, albeit at a significant computational cost. Alternatively, one can view collision response as an instantaneous reaction (an impulse), i.e., an instantaneous adjustment of the velocities. However, such adjustments are often problematic in the case of multiple contacts, as these may lead to a cyclic "trembling" behavior.

Our method belongs to a large family of *constraint-based* methods, which are increasingly the standard approach to contact handling. This set of methods meets our goals of providing robustness and improving efficiency of contact response, while minimizing the impact on the physics of the system.

[19] start from Signorini's law and derive the contact force formulation. The resulting equation is an LCP that is solved by Gauss–Seidel like iterations, sequentially resolving contacts until reaching the contact free state. [34] focus on robust treatment of collision without simulation artifacts. To enforce the no-collision constraint, this work uses an impulse response that gives rise to an LCP problem for its magnitude. To reduce the computational cost, the LCP solution (the Lagrange multiplier) is approximated by solving a linear system. [69] uses a linear approximation to contact constraints and a semi-implicit discretization, solving a mixed

LCP problem at each iteration.

Our approach is directly based on [36] and is closest to [2], in which the intersection volume and its gradient with respect to control vertices are computed at the candidate step. The non-collision is enforced as a constraint on this volume, which lead to a much smaller system compared to distance formulation between geometric primitives. The constrained formulation leads to an LCP problem. [36] assumes linear trajectory between edits and define space-time interference volume and uses it as a gap function and we use similar formulation to define the interference volume.

2.1.4 Nomenclature

In Table 2.1 we list symbols and operators used in this chapter. Throughout this chapter, lower case letters refer to scalars, and lowercase bold letters refer to vectors. Discretized quantities are denoted by sans serif letters.

2.2 Formulation

We start this section by stating the equations governing the flow in differential form and the imposed boundary conditions in Section 2.2.1. We introduce the requirements for the contact function, V, and its definition in Section 2.2.2. In Section 2.2.3, we impose no-contact as a constraint $V \geq 0$ to the differential equations introduced in Section 2.2.1 and derive the constrained formulation for the evolution equations. The set of integro-differential equations with contact constraint are solved using boundary integral formulation, which we outline in Section 2.2.4. The boundary integral formulation covers the cases for flows due to the

Symbol	Definition
γ_i	The boundary of the i^{th} vesicle
γ	$\cup_i \gamma_i$
μ	Viscosity of the ambient fluid
μ_i	Viscosity of the fluid inside i^{th} vesicle
ν_i	The viscosity contrast μ_i/μ
π_j	The boundary of the j^{th} rigid particle
π	$\cup_j \pi_j$
σ	Tension
χ	Shear rate
ϱ_i	The domain enclosed by γ_i
ϱ	$\cup_i \varrho_i$
$\mathcal{G}$	Stokes Single-layer operator
$\mathcal{T}$	Stokes Double-layer operator
LCP	Linear Complementarity problem
NCP	Nonlinear Complementarity Problem
SDC	Spectral Deferred Correction
STIV	Space-Time Interference Volumes

Symbol	Definition
LI	Locally-implicit time-stepping
CLI	Locally-implicit *constrained* time-stepping
GI	Globally-implicit time-stepping
d	Separation distance of particles
d_m	Minimum separation distance
$\boldsymbol{f}_\sigma$	Tensile force
$\boldsymbol{f}_b$	Bending force
$\boldsymbol{f}_c$	Collision force
h	Arclength distance between two discretization points
J	Jacobian of contact volumes V
$\boldsymbol{n}$	Unit outward normal
$\boldsymbol{u}$	Velocity
$\boldsymbol{u}^\infty$	The background velocity field
V	Contact volumes
$\boldsymbol{X}$	Coordinate of a (Lagrangian) point on a surface

Table 2.1: Index of frequently used symbols, operators, and abbreviations.

Dirichlet boundary condition on the fixed boundaries, moving rigid particles, and elastic vesicle membranes. The formulation in Section 2.2.4 follows the standard approach of potential theory [42, 77] and is presented in a concise manner.

2.2.1 Differential formulation

We consider the Stokes flow with N_v vesicles and N_p rigid particles suspended in a Newtonian fluid which is either confined or fills the free space, Fig. 2.1. In Stokesian flows, due to high viscosity and/or small length scale, the ratio of inertial and viscous forces (The Reynolds number) is small and the fluid flow can be described by the incompressible Stokes equation

$$-\mu\Delta\boldsymbol{u}(\boldsymbol{x}) + \nabla p(\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x}), \quad \text{and} \quad \nabla\cdot \boldsymbol{u}(\boldsymbol{x}) = 0 \qquad (\boldsymbol{x} \in \Omega), \tag{2.2.1}$$

$$\boldsymbol{F}(\boldsymbol{x}) = \int_\gamma \boldsymbol{f}(\boldsymbol{X})\delta(\boldsymbol{x} - \boldsymbol{X})\, \mathrm{d}s(\boldsymbol{X}), \tag{2.2.2}$$

where $\boldsymbol{f}$ is the surface density of the force exerted by the vesicle's membrane on the fluid and δ is the two-dimensional Dirac delta. The surface integral in Eq. (2.2.2) implies that $\boldsymbol{F}(\boldsymbol{x})$ is a distribution in the direction perpendicular to the surface. Ω denotes the fluid domain of interest with Γ_0 as its enclosing boundary (if present) and μ denoting the viscosity of ambient fluid. If Ω is multiply-connected, its interior boundary consists of K smooth curves denoted by $\Gamma_1, \dots, \Gamma_K$. The outer boundary Γ_0 encloses all the other connected components of the domain. The boundary of the domain is then denoted $\Gamma := \bigcup_k \Gamma_k$. We use $\boldsymbol{x}$ to denote an Eulerian point in the fluid ($\boldsymbol{x} \in \Omega$) and $\boldsymbol{X}$ a Lagrangian point on the vesicles or rigid particles. We let γ_i denote the boundary of the i^{th} vesicle ($i = 1, \dots, N_v$), ϱ_i denote the domain enclosed by γ_i, μ_i denote viscosity of the fluid inside that vesicle, and $\gamma := \bigcup_i \gamma_i$. Equation (2.2.1) is valid for $\boldsymbol{x} \in \varrho_i$ by replacing μ with μ_i.

There are rigid particles suspended in the fluid domain. We denote the boundary of the j^{th} rigid particle by π_j ($j = 1, \dots, N_p$) and let $\pi := \bigcup_j \pi_j$. The governing equations are augmented with the no-slip boundary condition on the surface of vesicles and particles

$$\boldsymbol{u}(\boldsymbol{X}, t) = \boldsymbol{X}_t \qquad (\boldsymbol{X} \in \gamma \cup \pi), \tag{2.2.3}$$

where $\boldsymbol{X}_t := \frac{\partial \boldsymbol{X}}{\partial t}$ is the material velocity of point $\boldsymbol{X}$ on the surface of vesicles or

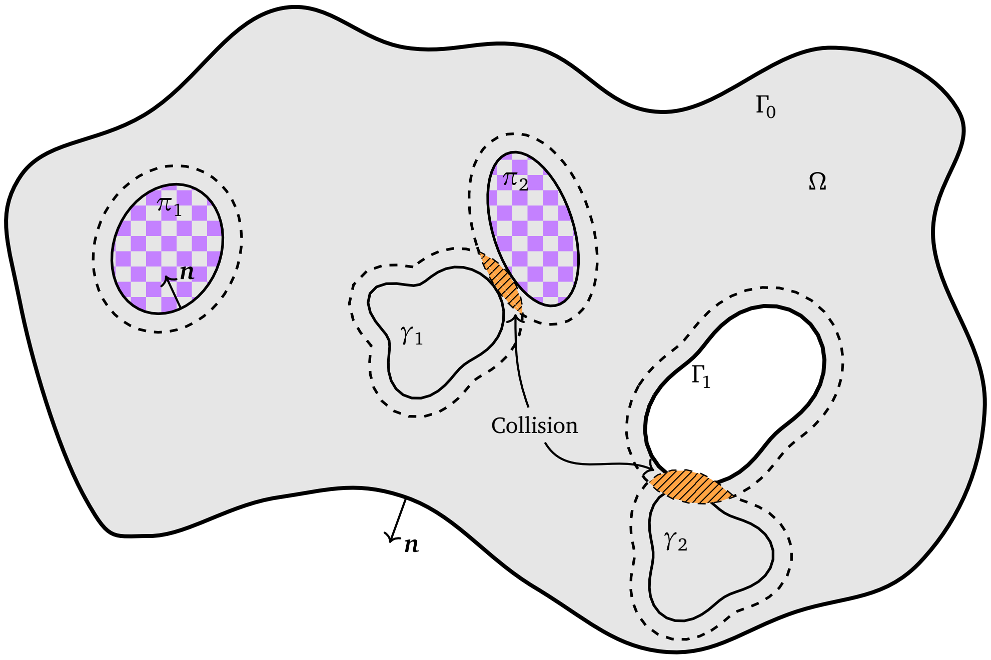


Figure 2.1: SCHEMATIC. *The flow domain Ω (gray shaded area) with boundary Γ_k $(k = 0, \dots, K)$. Vesicles and rigid particles are suspended in the fluid. The vesicle boundaries are denoted by γ_i $(i = 1, \dots, N_v)$ and the rigid particles (checkered pattern) are denoted by π_j $(j = 1, \dots, N_p)$. The outward normal vector to the boundaries is denotes by $\boldsymbol{n}$. The dotted lines around boundaries denote the prescribed minimum separation distance for each of them. The minimum separation distance is a parameter and can be set to zero. In this schematic, vesicle γ_1 and particle π_2 as well as vesicle γ_2 and boundary Γ_1 are in contact. The slices of the space-time intersection volumes at the current instance are marked by orange area.*

particles. The velocity on the fixed boundaries is imposed as a Dirichlet boundary condition

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{U}(\boldsymbol{x}) \qquad (\boldsymbol{x} \in \Gamma). \tag{2.2.4}$$

We assume that the vesicle membrane is inextensible, i.e.,

$$\boldsymbol{X}_s \cdot \boldsymbol{u}_s = 0 \qquad (\boldsymbol{X} \in \gamma), \tag{2.2.5}$$

where the subscript "s" denotes differentiation with respect to the arclength on the surface of vesicles.

Rigid particles are typically force- and torque-free. However, surface forces may be exerted on them due to a constraint, e.g., the contact force $\boldsymbol{f}_c$, which we will define later. In this case, the force $\boldsymbol{F}_j^\pi$ and torque L_j^π exerted on the j^{th} particle are the sum of such terms induced by constraints

$$\begin{aligned} &\boldsymbol{F}_j^\pi = \boldsymbol{0}, \quad \text{or} \quad \boldsymbol{F}_j^\pi = \int_{\pi_j} \boldsymbol{f}_c(\boldsymbol{X})\,\mathrm{d}s(\boldsymbol{X}) \qquad (j = 1, \dots, N_p), \\ &L_j^\pi = \boldsymbol{0}, \quad \text{or} \quad L_j^\pi = \int_{\pi_j} (\boldsymbol{X} - \boldsymbol{c}_j^\pi) \cdot \boldsymbol{f}_c{}^{\perp}(\boldsymbol{X})\,\mathrm{d}s(\boldsymbol{X}) \qquad (j = 1, \dots, N_p), \end{aligned} \tag{2.2.6}$$

where $\boldsymbol{c}_j^\pi$ is the center of mass for π_j and $\boldsymbol{f}^\perp = (f_1, f_2)^\perp := (f_2, -f_1)$.

2.2.2 Contact definition

It is known [27, 65] that the exact solution of equations of motion, Eqs. (2.2.1), (2.2.3), and (2.2.4), keeps particles apart in finite time due to formation of lubrication film. Thus, it is theoretically sufficient to solve the equations with an adequate degree of accuracy to avoid any problems related to overlaps between particles. Nonetheless, achieving this accuracy for many types of flows (most notably, flows with high volume fraction of particles or with complex boundaries) is prohibitively expensive.

With inadequate computational accuracy particles may intersect with each other or with the boundaries and depending on the numerical method used, the consequences of this varies. For methods based on integral equations the consequences are particularly dramatic, as overlapping boundaries lead to divergent integrals. To address this issue, we augment the governing equations with a contact constraint, formally written as

$$V(\boldsymbol{u}, t) \geq 0, \tag{2.2.7}$$

The function V is chosen in a way that $V < 0$ implies some parts of the surface $S = \Gamma \cup \gamma \cup \pi$ are at a distance less than a user-specified constant d_m. Function V may be a vector-valued function, for which the inequality is understood component-wise. This constraint ensures that the suspension remains contact-free independent of the numerical resolution.

For the constraint function V, in addition to the basic condition above, we choose a function that satisfies these additional criteria:

(i) it introduces a relatively small number of additional constraints, and

(ii) when the function is discretized, no contact is missed even for large time step.

To clarify the second condition, suppose we have a small particle rapidly moving towards a planar boundary. For a large time step, it may move to the other side of the boundary in a single step, so any condition that considers an instantaneous quantity depending on only the current position is likely to miss such contact.

To this end, we extend the *Space-Time Interference Volumes* (STIV) from [36] to define the function V^C as the area in space-time swept by the intersecting segments of the boundary over time. To be more precise, for each point $\boldsymbol{X}(s, t_0)$ on the boundary, consider a trajectory $\boldsymbol{X}(s, \tau)$, between a time t_0, for which there are no collisions, and a time t. Points $\boldsymbol{X}(s, \tau)$ define a deformed boundary $S(\tau)$ for each τ. For each point $\boldsymbol{X}(s, \tau)$, we define $\tau_I(s)$, $t_0 \leq \tau_I \leq t$, to be the first instance for which this point comes into contact with a different point of $S(\tau_I)$. Assuming an interference-free configuration at t_0, the space-time volume constraint for the time interval $[t_0, t]$ is

$$V^C(S,t) = -\int_{S(t_0)} \int_{\tau_I(s)}^{t} \sqrt{\epsilon^2 + (\boldsymbol{X}_t(s,\tau)\cdot \boldsymbol{n}(s,\tau))^2}\,\mathrm{d}\tau\,\mathrm{d}s, \tag{2.2.8}$$

where $\boldsymbol{n}(s,\tau)$ denotes the normal to $S(\tau)$ at $\boldsymbol{X}(s,\tau)$. The integration is over all points for which $\tau_I(s) \in [t_0, t]$. For two dimensional flows, it is the area of the surface formed by the points in 3D with coordinate $(\boldsymbol{X}(s,\tau), \epsilon\tau)$, for all (s,τ) such that $\tau_I(s) \le t$. To arrive at this formula, we used the fact that the surface is inextensible and thus the surface metric does not change.

This is a modified continuous version of the discrete functional described in [36]. The functional used in Harmon et al. differs in the following respects: (i) it is defined for piecewise linear trajectories directly; (ii) $\epsilon = 0$; (iii) the normal is taken to be the vector at the time of contact $\boldsymbol{n}(s, \tau_I(s))$. In practice we observe little difference in the behavior of two functionals, we choose this formulation as corresponds directly to the space-time volume. The version of Harmon et al. can be visualized as projection of the space-time volume to the spatial plane. The constant ϵ we introduce, which has units of velocity, effectively replaces $|\boldsymbol{u}\cdot\boldsymbol{n}|$ with $\sqrt{\epsilon^2 + (\boldsymbol{u}\cdot\boldsymbol{n})^2}$ in the original formulation, smoothing out the constraint expression. Another important property of this choice of function, compared to, e.g., a space intersection volume, is that for even a very thin object moving at high velocity, it will be proportional to the time interval $t - t_0$.

Infinitesimal version of the constraint Consider the constraint given in Eq. (2.2.8) on the interval $[t, t+\Delta t]$, where the configuration is collision-free at time t. For a fixed τ, the contact area, i.e., the set of points s such that $\tau_I(s) \le \tau$, defines a set of boundary segments. We consider one such segment as a contact zone and let $s_1(\tau)$ and $s_2(\tau)$ be the extents of such a contact zone at time τ. We rewrite the STIV integral for this contact zone by exchanging the order of integration, using $\tau(s) \leq \tau$ is equivalent to $s_1(\tau) \leq s \leq s_2(\tau)$:

$$\Delta V^C = -\int_t^{t+\Delta t} \int_{s_1(\tau)}^{s_2(\tau)} \sqrt{\epsilon^2 + (\boldsymbol{u}(s,\tau) \cdot \boldsymbol{n}(s,\tau))^2}\, \mathrm{d}s\, \mathrm{d}\tau. \tag{2.2.9}$$

Neglecting higher-order terms in Δt, we obtain:

$$\Delta V^C = -\int_{s_1(t)}^{s_2(t)} \sqrt{\epsilon^2 + (\boldsymbol{u}(s,t) \cdot \boldsymbol{n}(s,t))^2}\, \mathrm{d}s\ \Delta t, \tag{2.2.10}$$

implying the rate of change of the space-time volume with respect to time.

As the maximal value of the integrand is $-\epsilon$, we add ϵ to make sure that the constraint can be zero, defining

$$V(\boldsymbol{u}, t) = -\int_{s_1(t)}^{s_2(t)} \sqrt{\epsilon^2 + (\boldsymbol{u}(s,t) \cdot \boldsymbol{n}(s,t))^2}\, \mathrm{d}s + \epsilon, \tag{2.2.11}$$

which we will use in the next section as a constraint for the fluid flow. The variation of this constraint with respect to $\boldsymbol{u}$ is

$$\mathrm{d}_{\boldsymbol{u}} V[\delta \boldsymbol{u}] = -\int_{s_1(t)}^{s_2(t)} \frac{(\boldsymbol{n} \cdot \boldsymbol{u})(\boldsymbol{n} \cdot \delta \boldsymbol{u})}{\sqrt{\epsilon^2 + (\boldsymbol{u} \cdot \boldsymbol{n})^2}}\, \mathrm{d}s. \tag{2.2.12}$$

We consider each connected component of this (infinitesimal) volume as a separate volume, and impose an inequality constraint on each; while keeping a single volume is in principle equivalent, using multiple volumes avoid certain undesirable effects in discretization [36]. Thus, $V(\boldsymbol{u}, t)$ is a vector function of time-dependent dimension, with one component per active contact region.

Depending on the context, we may omit the dependence of V on $\boldsymbol{u}$ and write $V(t)$ as the contact volume function or $V(\gamma_i, t)$ for elements of $V(\boldsymbol{u}, t)$ involving surface γ_i.

In practice, it is desirable to control the minimal distance between particles. Therefore, we define a minimum separation distance $d_m \geq 0$ and modify the constraint such that particles are in contact when they are within d_m distance from each other; as shown in Fig. 2.1. The contact volume with minimum separation distance is calculated with the surface displaced by d_m, i.e., the time t_I or, equivalently, the contact segment $[s_1, s_2]$ is obtained not from the first contact with $S(\tau)$ but rather the displaced surface $S(\tau) + d_m\boldsymbol{n}(\tau)$. Maintaining minimum separation distance — rather than considering pure contact only — eliminates of potentially expensive computation of nearly singular integrals close to the surface and improves the accuracy in semi-explicit time-stepping.

2.2.3 Contact constraint

We use the Lagrange multiplier method (e.g., [110]) to add contact constraints, Eq. (2.2.11), to the system. While it is computationally more expensive than adding a penalty force for the constraint (effectively, an artificial repulsion force), it has the advantage of eliminating the need of tuning the parameters of the penalty force to ensure that the constraint is satisfied and keeping nonphysical forces introduced into the system to the minimum required for maintaining the desired separation. The constrained system can be written as

$$\min \int_{\Omega} \left(\frac{1}{2}\mu \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \boldsymbol{u} \cdot \boldsymbol{F} \right) \mathrm{d}A, \tag{2.2.13}$$

$$\begin{aligned} \text{subject to:} \qquad & \nabla \cdot \boldsymbol{u}(\boldsymbol{x}) = 0 \quad (\boldsymbol{x} \in \Omega), \\ & \boldsymbol{X}_s \cdot \boldsymbol{u}_s = 0 \quad (\boldsymbol{X} \in \gamma), \\ & V(\boldsymbol{u}, t) \geq 0. \end{aligned}$$

If we omit the inequality constraint, the remaining three equations are equivalent to the Stokes equations (2.2.1). Since we are solving a quasi-static system where the PDE is elliptic and the system is evolved due to no-slip boundary condition, the system is in force balance at all instances. Also, the contact constraint is in fact on the velocity field that evolves the surface. The Lagrangian for this system is

$$\mathcal{L}(\boldsymbol{u}, p, \sigma, \lambda) = \int_{\Omega} \left(\frac{1}{2}\mu \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \boldsymbol{u} \cdot \boldsymbol{F} - p \nabla \cdot \boldsymbol{u} \right) \mathrm{d}A + \int_{\gamma} \sigma \boldsymbol{X}_s \cdot \boldsymbol{u}_s \, \mathrm{d}s + V \cdot \lambda. \tag{2.2.14}$$

The first-order optimality (KKT) conditions yield the following modified Stokes equation, along with the constraints listed in Eq. (2.2.13):

$$-\mu\Delta\boldsymbol{u} + \nabla p = \boldsymbol{F}', \tag{2.2.15}$$

$$\boldsymbol{F}'(\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x}) + \int_\gamma \boldsymbol{f}_\sigma \delta(\boldsymbol{x}-\boldsymbol{X})\,\mathrm{d}s + \int_S \boldsymbol{f}_c \delta(\boldsymbol{x}-\boldsymbol{X})\,\mathrm{d}s, \tag{2.2.16}$$

$$\boldsymbol{f}_\sigma = -(\sigma\boldsymbol{X}_s)_s, \tag{2.2.17}$$

$$\boldsymbol{f}_c = \mathrm{d}_{\boldsymbol{u}}V^T\lambda, \tag{2.2.18}$$

$$\lambda \geq 0, \tag{2.2.19}$$

$$\lambda \cdot V = 0, \tag{2.2.20}$$

where the last condition is the complementarity condition — either an equality constraint is active ($V_i = 0$) or its corresponding Lagrange multiplier λ_i is zero. As we will see in the next section and based on Eq. (2.2.16), the collision force $\boldsymbol{f}_c$ is added to the traction jump across the vesicle's interface. For rigid particles, the contact force induces force and torque on each particle — as given in Eq. (2.2.6).

It is customary to combine $V \geq 0, \lambda \geq 0$, and $\lambda \cdot V = 0$, into one expression and write

$$0 \leq V(t) \quad \perp \quad \lambda \geq 0, \tag{2.2.21}$$

where "$\perp$" denotes the complementarity condition. These ensure that the Signorini conditions introduced in Section 3.1 are respected: contacts do not produce attraction force ($\lambda \geq 0$) and the constraint is active (λ nonzero) if and only if $V(t)$ is zero. We observe from Eq. (2.2.12) that for admissible velocities normal to the contact, $\mathrm{d}_{\boldsymbol{u}}V[\delta\boldsymbol{u}]$ is zero for any $\delta\boldsymbol{u}$. Therefore in the smooth case, the force $\mathrm{d}_{\boldsymbol{u}}V^T\lambda$ does no work.

2.2.4 Boundary integral formulation

Following the standard approach of potential theory [74, 77], one can express the solution of the Stokes boundary value problem, Eq. (2.2.15), as a system of singular integro-differential equations on all immersed and bounding surfaces. Here, we outline general formulae that we use in our framework and refer the interested reader to [42, 77] for in depth treatments of the subject.

The Stokeslet tensor $\mathbf{G}$, the Stresslet tensor $\mathbf{T}$, and the Rotlet $\mathbf{R}$ are the fundamental solutions of the Stokes equation and are given by

$$\mathbf{G}(\boldsymbol{r}) = \frac{1}{4\pi\mu}\left(-\log\|\boldsymbol{r}\|\boldsymbol{I} + \frac{\boldsymbol{r}\otimes\boldsymbol{r}}{\|\boldsymbol{r}\|^2}\right), \tag{2.2.22}$$

$$\mathbf{T}(\boldsymbol{r}) = \frac{1}{\pi}\frac{\boldsymbol{r}\otimes\boldsymbol{r}\otimes\boldsymbol{r}}{\|\boldsymbol{r}\|^4}, \tag{2.2.23}$$

$$\mathbf{R}(\boldsymbol{r}) = \frac{1}{4\pi\mu}\frac{\boldsymbol{r}^\perp}{\|\boldsymbol{r}\|^2}, \tag{2.2.24}$$

where $\boldsymbol{r}^\perp = (r_1, r_2)^\perp := (r_2, -r_1)$ and $\otimes$ denotes the tensor product.

The solution of Eq. (2.2.15) can be expressed by the combination of single- and double-layer integrals. We denote the single-layer integral on the vesicle surface γ_i by

$$\mathcal{G}_{\gamma_i}[\boldsymbol{f}](\boldsymbol{x}) := \int_{\gamma_i} \mathbf{G}(\boldsymbol{x}-\boldsymbol{Y})\cdot\boldsymbol{f}(\boldsymbol{Y})\,\mathrm{d}s(\boldsymbol{Y}), \tag{2.2.25}$$

where $\boldsymbol{f}$ is an appropriately defined density. The double-layer integral on a surface S (a vesicle, a rigid particle, or a fixed boundary) is

$$\mathcal{T}_S[\boldsymbol{q}](\boldsymbol{x}) := \int_S \boldsymbol{n}(\boldsymbol{Y})\cdot\mathbf{T}(\boldsymbol{x}-\boldsymbol{Y})\cdot\boldsymbol{q}(\boldsymbol{Y})\,\mathrm{d}s(\boldsymbol{Y}), \tag{2.2.26}$$

where $\boldsymbol{n}$ denotes the outward normal to the surface S (as shown in Fig. 2.1), and

$\boldsymbol{q}$ is an appropriately defined density. When the evaluation point $\boldsymbol{x}$ is on the integration surface, Eq. (2.2.25) is a singular integral, and Eq. (2.2.26) is interpreted in the principal value sense.

Due to the linearity of the Stokes equations, as formulated in [82, 86], the velocity at a point $\boldsymbol{x} \in \Omega$ can be expressed as the superposition of velocities due to vesicles, rigid particles, and fixed boundaries

$$\alpha \boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}^{\infty}(\boldsymbol{x}) + \boldsymbol{u}^{\gamma}(\boldsymbol{x}) + \boldsymbol{u}^{\pi}(\boldsymbol{x}) + \boldsymbol{u}^{\Gamma}(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \quad \alpha = \begin{cases} 1 & \boldsymbol{x} \in \Omega \backslash \varrho, \\ \nu_i & \boldsymbol{x} \in \varrho_i, \\ (1+\nu_i)/2 & \boldsymbol{x} \in \gamma_i, \end{cases} \tag{2.2.27}$$

where $\boldsymbol{u}^{\infty}(\boldsymbol{x})$ represent the background velocity field (for unbounded flows) and $\nu_i = \mu_i/\mu$ denotes the viscosity contrast of the i^{th} vesicle. The velocity contributions from vesicles, rigid particles, and fixed boundaries each can be further decomposed into the contribution of individual components

$$\boldsymbol{u}^{\gamma}(\boldsymbol{x}) = \sum_{i=1}^{N_v} \boldsymbol{u}_i^{\gamma}(\boldsymbol{x}), \qquad \boldsymbol{u}^{\pi}(\boldsymbol{x}) = \sum_{j=1}^{N_p} \boldsymbol{u}_j^{\pi}(\boldsymbol{x}), \qquad \boldsymbol{u}^{\Gamma}(\boldsymbol{x}) = \sum_{k=0}^{K} \boldsymbol{u}_k^{\Gamma}(\boldsymbol{x}). \tag{2.2.28}$$

To simplify the representation, we introduce the complementary velocity for each boundary component, as a shorthand to denote the velocity field induced by other particles at point $\boldsymbol{x}$. For the i^{th} vesicle, it is defined as $\bar{\boldsymbol{u}}_i^{\gamma} = \alpha \boldsymbol{u} - \boldsymbol{u}_i^{\gamma}$. The complementary velocity is defined in a similar fashion for rigid particles as well as components of the fixed boundary.

2.2.4.1 The contribution from vesicles

The velocity induced by the i^{th} vesicle is expressed as an integral [77]:

$$\boldsymbol{u}_i^{\gamma}(\boldsymbol{x}) = \mathcal{G}_{\gamma_i}[\boldsymbol{f}](\boldsymbol{x}) + (1-\nu_i)\mathcal{T}_{\gamma_i}[\boldsymbol{u}](\boldsymbol{x}) \qquad (\boldsymbol{x} \in \Omega), \tag{2.2.29}$$

where the double-layer density $\boldsymbol{u}$ is the total interface velocity and $\boldsymbol{f}$ is the traction jump across the vesicle membrane [103]. Based on Eq. (2.2.16), the traction jump is equal to the sum of bending, tensile, and collision forces (when present)

$$\boldsymbol{f}(\boldsymbol{X}) = \boldsymbol{f}_b + \boldsymbol{f}_\sigma + \boldsymbol{f}_c = -\kappa_b \boldsymbol{X}_{ssss} - (\sigma \boldsymbol{X}_s)_s + \mathrm{d}_{\boldsymbol{u}} V^T \lambda \qquad (\boldsymbol{X} \in \gamma), \tag{2.2.30}$$

where κ_b is the membrane's bending modulus. The tensile force $\boldsymbol{f}_\sigma = (\sigma \boldsymbol{X}_s)_s$ is determined by the local inextensibility constraint, Eq. (2.2.5), and the tension σ is its Lagrangian multiplier, Eq. (2.2.17).

Note that Eq. (2.2.29) is the contribution from each vesicle to the velocity field. To obtain an equation for the interfacial velocity, Eq. (2.2.29) is to be substituted into Eq. (2.2.27) and evaluated at $\boldsymbol{X} \in \gamma_i$:

$$\frac{(1+\nu_i)}{2}\boldsymbol{u}(\boldsymbol{X}) = \bar{\boldsymbol{u}}_i^{\gamma}(\boldsymbol{X}) + \mathcal{G}_{\gamma_i}[\boldsymbol{f}](\boldsymbol{X}) + (1-\nu_i)\mathcal{T}_{\gamma_i}[\boldsymbol{u}](\boldsymbol{X}) \qquad (\boldsymbol{X} \in \gamma_i), \tag{2.2.31}$$

subject to the local inextensibility constraint

$$\boldsymbol{X}_s \cdot \boldsymbol{u}_s = 0 \qquad (\boldsymbol{X} \in \gamma_i). \tag{2.2.32}$$

2.2.4.2 The contribution from the fixed boundaries

The velocity contribution from the fixed boundary can be expressed as a double-layer integral [74] along Γ. The contribution of the outer boundary Γ_0 is

$$\boldsymbol{u}_0^\Gamma(\boldsymbol{x}) = \mathcal{T}_{\Gamma_0}[\boldsymbol{\eta_0}](\boldsymbol{x}) \qquad (\boldsymbol{x} \in \Omega), \tag{2.2.33}$$

where $\boldsymbol{\eta}_0$ is the density to be determined based on boundary conditions. Substituting Eq. (2.2.33) into Eq. (2.2.27) and taking its limit to a point on Γ_0 and using the Dirichlet boundary condition, Eq. (2.2.4), we obtain a Fredholm integral equations for the density $\boldsymbol{\eta}_0$

$$\boldsymbol{U}(\boldsymbol{x}) - \bar{\boldsymbol{u}}_0^\Gamma(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{\eta}_0(\boldsymbol{x}) + \mathcal{T}_{\Gamma_0}[\boldsymbol{\eta_0}](\boldsymbol{x}) \qquad (\boldsymbol{x} \in \Gamma_0).$$

However, this equation is rank deficient [40]. To render it invertible, the equation is modified following [40]:

$$\boldsymbol{U}(\boldsymbol{x}) - \bar{\boldsymbol{u}}_0^\Gamma(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{\eta}_0(\boldsymbol{x}) + \mathcal{T}_{\Gamma_0}[\boldsymbol{\eta_0}](\boldsymbol{x}) + \mathcal{N}_{\Gamma_0}[\boldsymbol{\eta_0}](\boldsymbol{x}) \qquad (\boldsymbol{x} \in \Gamma_0), \tag{2.2.34}$$

where the operator $\mathcal{N}_{\Gamma_0}$ is defined as

$$\mathcal{N}_{\Gamma_0}[\boldsymbol{\eta_0}](\boldsymbol{x}) = \int_{\Gamma_0} [\boldsymbol{n}(\boldsymbol{x}) \otimes \boldsymbol{n}(\boldsymbol{y})] \cdot \boldsymbol{\eta_0}(\boldsymbol{y}) \, \mathrm{d}s(\boldsymbol{y}) \qquad (\boldsymbol{x} \in \Gamma_0). \tag{2.2.35}$$

For the enclosed boundary components Γ_k $(k > 0)$, to eliminate the double-layer nullspace we need to include additional Stokeslet and Rotlet terms

$$\boldsymbol{u}_k^\Gamma(\boldsymbol{x}) = \mathcal{T}_{\Gamma_k}[\boldsymbol{\eta_k}](\boldsymbol{x}) + \mathbf{G}(\boldsymbol{x} - \boldsymbol{c}_k^\Gamma) \cdot \boldsymbol{F}_k^\Gamma + \mathbf{R}(\boldsymbol{x} - \boldsymbol{c}_k^\Gamma) L_k^\Gamma, \qquad (k = 1, \ldots, K; \boldsymbol{x} \in \Omega), \tag{2.2.36}$$

where $\boldsymbol{c}_k^\Gamma$ is a point enclosed by Γ_k, $\boldsymbol{F}_k^\Gamma$ is the force exerted on Γ_k, and L_k^Γ is the torque:

$$\boldsymbol{F}_k^\Gamma = \frac{1}{|\Gamma_k|} \int_{\Gamma_k} \boldsymbol{\eta_k} \, \mathrm{d}s, \qquad L_k^\Gamma = \frac{1}{|\Gamma_k|} \int_{\Gamma_k} (\boldsymbol{X} - \boldsymbol{c}_k^\Gamma) \cdot \boldsymbol{\eta}_k^\perp \, \mathrm{d}s, \tag{2.2.37}$$

where $|\Gamma_k|$ denotes the perimeter of Γ_k. Taking the limit to points on the surface Γ_k, leads to the following integral equation:

$$\boldsymbol{U}(\boldsymbol{x}) - \bar{\boldsymbol{u}}_k^\Gamma(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{\eta}_k(\boldsymbol{x}) + \mathcal{T}_{\Gamma_k}[\boldsymbol{\eta_k}](\boldsymbol{x}) + \mathbf{G}(\boldsymbol{x} - \boldsymbol{c}_k^\Gamma) \cdot \boldsymbol{F}_k^\Gamma + \mathbf{R}(\boldsymbol{x} - \boldsymbol{c}_k^\Gamma) L_k^\Gamma \quad (\boldsymbol{x} \in \Gamma_k). \tag{2.2.38}$$

Equations (2.2.37) and (2.2.38) are a complete system for double-layer densities $\boldsymbol{\eta}_k$, forces $\boldsymbol{F}_k^\Gamma$, and torques L_k^Γ on each surface Γ_k.

2.2.4.3 The contribution from rigid particles

The formulation for rigid particles is very similar to that of fixed boundaries, except the force and torque are known — cf. Eq. (2.2.6). The velocity contribution from the j^{th} rigid particle is

$$\boldsymbol{u}_j^\pi(\boldsymbol{x}) = \mathcal{T}_{\pi_j}[\boldsymbol{\zeta}_j](\boldsymbol{x}) + \boldsymbol{G}(\boldsymbol{x} - \boldsymbol{c}_j^\pi) \cdot \boldsymbol{F}_j^\pi + \boldsymbol{R}(\boldsymbol{x} - \boldsymbol{c}_j^\pi) L_j^\pi, \tag{2.2.39}$$

Where $\boldsymbol{F}_j^\pi, L_j^\pi$ are, respectively, the *known* net force and torque exerted on the particle and $\boldsymbol{\zeta}_j$ is the unknown density.

Let $\boldsymbol{U}_j^\pi$ and ω_j^π be the translational and angular velocities of the j^{th} particle; then we obtain the following integral equation for the density $\boldsymbol{\zeta}_j$ from the limit of (2.2.39):

$$\boldsymbol{U}_j^\pi+\omega_j^\pi(\boldsymbol{X}-\boldsymbol{c}_j^\pi)^\perp-\bar{\boldsymbol{u}}_j^\pi(\boldsymbol{X}) = -\frac{1}{2}\boldsymbol{\zeta}_j(\boldsymbol{X})+\mathcal{T}_{\pi_j}[\boldsymbol{\zeta}_j](\boldsymbol{X})+\boldsymbol{G}(\boldsymbol{X}-\boldsymbol{c}_j^\pi)\cdot\boldsymbol{F}_j^\pi+\boldsymbol{R}(\boldsymbol{X}-\boldsymbol{c}_j^\pi)L_j^\pi. \tag{2.2.40}$$

where

$$\boldsymbol{F}_j^\pi = \frac{1}{|\pi_j|}\int_{\pi_j}\boldsymbol{\zeta}_j\,\mathrm{d}s, \qquad L_j^\pi = \frac{1}{|\pi_j|}\int_{\pi_j}(\boldsymbol{Y}-\boldsymbol{c}_j^\pi)\cdot\boldsymbol{\zeta}_j^\perp\,\mathrm{d}s \tag{2.2.41}$$

where $\boldsymbol{c}_j^\pi$ is the center of j^{th} rigid particle. Equations (2.2.40) and (2.2.41) are used to solve for the unknown densities $\boldsymbol{\zeta}_j$ as well as the unknown translational and angular velocities of each particle. Note that the objective of Eqs. (2.2.37) and (2.2.41) is to remove the null space of the double-layer operator and therefore their left-hand-side (i.e., the projection of the solution onto the null space) can be chosen rather arbitrarily.

2.2.5 Formulation summary

The formulae outlined above govern the evolution of the suspension. The flow constituents are hydrodynamically coupled through the complementary velocity. Given the configuration of the suspension, the unknowns are:

- Velocity $\boldsymbol{u}(\boldsymbol{X})$ and tension σ of vesicles' interface determined by Eqs. (2.2.30–2.2.32). The velocity is integrated for the vesicles' trajectory using Eq. (2.2.3).

- The double-layer density on the enclosing boundary $\boldsymbol{\eta}_0$ as well as the double-layer density $\boldsymbol{\eta}_k$ ($k = 1, \dots, K$), force $\boldsymbol{F}_k^\Gamma$, and torque L_k^Γ on the interior boundaries determined by Eqs. (2.2.34), (2.2.37), and (2.2.38). Note that the collision constraint does not enter the formulation for the fixed boundaries and when a particle collides with a fixed boundary, the collision force is only applied to the particle. The unknown force and torque above can be interpreted as the required force to keep the interior boundary piece stationary.

- Translational $\boldsymbol{U}_j^\pi$ and angular ω_j^π velocities of rigid particles ($j = 1, \dots, N_p$) as well as double-layer densities $\boldsymbol{\zeta}_j$ on their boundary determined by Eqs. (2.2.40) and (2.2.41). Where the force and torque are either zero or determined by the collision constraint Eq. (2.2.6).

This system is constrained by Signorini (KKT) conditions for the contact, Eq. (2.2.21), which is used to compute λ, the strength of the contact force.

In the referenced equations above, the complementary velocity is combination of velocities given in Eqs. (2.2.29), (2.2.33), (2.2.36), and (2.2.39).

Parameters and scaling The characteristic length for a system with elastic vesicles is defined as $R_0 = L/2\pi$ where L denotes the perimeter of a vesicle. The characteristic time is defined as $\tau = \mu L^3/\kappa_b$, where μ is the viscosity of the suspending fluid and κ_b is the vesicles' bending modulus. The reduced area for a vesicle is defined as $\frac{A}{\pi R_0^2}$. The reduced area is used extensively to classify vesicles' shape and dynamics. In the shear flows, the non-dimensional shear rate is defined as $\hat{\chi} = \tau\chi$, where τ is the characteristic time and χ is the shear rate. For other types of flow, local shear rate within the domain is used for scaling. Hereinafter, without change of notation, we use quantities non-dimensionalized by characteristic variables [103].

2.3 Discretization and Numerical Methods

In this section, we describe the numerical algorithms required for solving the dynamics of a particulate Stokesian suspension. We use the spatial representation and integral schemes presented in [86]. We also adapt the spectral deferred correction time-stepping from [83, 84] to the local implicit time-stepping schemes. Furthermore, we use piecewise-linear discretization of curves to calculate the space-time contact volume $V(\gamma, t)$, Eq. (2.2.8), similar to [36]. To solve the complementarity problem resulting from the contact constraint, we use the minimum-map Newton method discussed in [21] or [15, Section 5.8].

The key difference, compared to previous works on particulate suspensions is that at every time step instead of solving a linear system we solve a nonlinear complementarity problem (NCP). The NCPs are solved iteratively by recursive linearization and using a Linear Complementarity Problem (LCP) solver. We refer to these iterations as contact-resolving iterations, in contrast to the outer time-stepping iterations.

For simplicity, we describe the numerical scheme for a system including vesicles only, without boundaries or rigid particles. Adding these requires straightforward modifications to the equations. In the following sections, we will first summarize the spatial discretization, then discuss the LCP solver, and close with the time discretization with contact constraint.

2.3.1 Spatial discretization

All interfaces are discretized with N uniformly-spaced discretization points [86]. The number of points on each curve is typically different but for the sake of clarity we denote that number by N. The distance between discretization points over the curves does not change with time due to the rigidity of particles or the local inextensibility constraint for vesicles. Let $\boldsymbol{X}(s)$, with $s \in (0, L]$, be a parametrization of the interface γ_i (or π_j), and let $\{s_k = kL/N\}_{k=1}^{N}$ be N equally spaced points in arclength parameter, and $\mathbf{X}_k := \boldsymbol{X}(s_k)$ denote the corresponding material points.

High-order discretization for force computation We use the Fourier basis to interpolate the positions and forces associated with sample points, and FFT to calculate the derivatives of all orders to spectral accuracy. For computing surface integrals with smooth integrand, we use the composite trapezoidal rule that provides spectral accuracy. We use the hybrid Gauss-trapezoidal quadrature rules of [3] to integrate the singular single-layer potential for $\boldsymbol{X} \in \gamma_i$

$$\mathcal{G}_{\gamma_i}[\boldsymbol{f}](\boldsymbol{X}) \approx \mathbf{G}_{\gamma_i}[\mathbf{f}](\mathbf{X}) := \sum_{\ell=1}^{N+M} w_\ell \mathbf{G}(\mathbf{X} - \mathbf{Y}_\ell) \cdot \mathbf{f}(\mathbf{Y}_\ell), \tag{2.3.1}$$

where w_ℓ are the quadrature weights given in [3, Table 8] and $\mathbf{Y}_\ell$ are quadrature points. Collocating the integral equation on $\mathbf{X}$ the linear operator in Eq. (2.3.1) is a matrix that we denote by $\mathbf{G}_{\gamma_i}$.

The double-layer kernel $\boldsymbol{n}(\boldsymbol{Y}) \cdot \mathbf{T}(\boldsymbol{X} - \boldsymbol{Y})$ in Eq. (2.2.26) is non-singular in two dimensions

$$\lim_{\gamma_i \ni \boldsymbol{X} \to \boldsymbol{Y}} \boldsymbol{n}(\boldsymbol{Y}) \cdot \mathbf{T}(\boldsymbol{X} - \boldsymbol{Y}) = -\frac{\kappa}{2\pi} \boldsymbol{t} \otimes \boldsymbol{t},$$

where $\boldsymbol{t}$ denotes the tangent vector at $\boldsymbol{Y}$. Therefore, a simple uniform-weight composite trapezoidal quadrature rule has spectral accuracy in this case. Similar to the single-layer case, we denote the discrete double-layer operator on γ_i by $\mathbf{T}_{\gamma_i}$. We use the nearly-singular integration scheme described in [82] to maintain high integration accuracy for particles closely approaching each other.

Piecewise-linear discretization for constraints While the spectral spatial discretization is used for most computations, it poses a problem for the minimal-separation constraint discretization. Computing parametric curve intersections, an essential step in the STIV computation, is relatively expensive and difficult to implement robustly, as this requires solving nonlinear equations. We observe that the sensitivity to the separation distance on the overall accuracy is low in most situations, as explored in Section 2.4. Thus, rather than enforcing the constraint as precisely as allowed by the spectral discretization, we opt for a low-order, piecewise-linear discretization in this case, and use an algorithm that ensures that *at least* the target minimal separation is maintained, but may enforce a higher separation distance.

For the purpose of computing STIV and its gradient, we use $L(\mathbf{X}, r)$, the piecewise-linear interpolant of r times refinement of points — the refined points correspond to arclength values with spacing $L/(N2^r)$, with r determined adaptively.

For discretized computations, we set the separation distance to $(1 + 2\alpha)d_m$, where d_m is the target minimum separation distance. We choose r such that $\|L(\mathbf{X}, r) - \boldsymbol{X}\|_\infty < \alpha d_m$. Our NCP solver, described below, ensures that the separation between parts of $L(\mathbf{X}, r)$ is $(1 + 2\alpha)d_m$ at the end of a single time step. We choose $\alpha = 0.1$, which requires $r = 1$ in our experiments; smaller values of α

require more refinement and enforce the constraint more accurately.

At the end of the time step, the minimal-separation constraint ensures that $L(\mathbf{X}, r)(s)$, for any s, is at least at the distance $(1+2\alpha)d_m$ from a possible intersection if its trajectory is extrapolated linearly. By computing the upper bounds on the difference between the $\boldsymbol{X}(s)$ and $L(\mathbf{X}, r)(s)$ at the beginning of the time step, and interpolated velocities, we obtain a lower bound on the actual separation distance d' for the spectral surface $\boldsymbol{X}(s)$. If $d' < d_m$, we increase r, and repeat the time step. As the piecewise linear approximation converges to the spectral boundary $\boldsymbol{X}$, and so do the interpolated velocities. In practice, we have not observed a need for refinement for our choice of α.

Computing the contact constraint To discretize the constraint forces, rather than discretizing directly the infinitesimal functional Eq. (2.2.11), we closely follow the approach of [36], and compute the finite STIV via discretization of Eq. (2.2.8), approximating the motion of the vertices with piecewise linear functions, and computing times of intersections for each vertex separately. While resulting in more complex expressions versus direct discretization of Eq. (2.2.11), this approach results in less extreme changes in forces for larger time steps.

For the piecewise-linear discretization of curves, the space-time contact volume $V(\gamma, t)$, Eq. (2.2.8), and its gradient are calculated similar to the definitions and algorithms in [36]. Given a contact-free configuration and a candidate configuration for the next time step, we calculate the discretized space-time contact volume as the sum of edge-vertex contact volumes $V = \sum_k V_k(\mathbf{e}, \mathbf{X})$, where k indexes edge-vertex pairs. We use a regular spatial grid of size proportional to the average boundary spacing to quickly find potential collisions. For all vertices and edges, the bounding box enclosing their initial (collision-free) and final (candidate position) locations is formed and all the grid boxes intersecting that box are marked. When the minimal separation distance $d_m > 0$, the bounding box is enlarged by d_m. For each edge-vertex pair $\mathbf{e}(\mathbf{X}_i, \mathbf{X}_{i+1})$ and $\mathbf{X}_k$, we solve a quartic equation to find their earliest contact time τ_I assuming linear trajectory between initial and candidate, where the vertex velocity is defined as $\mathbf{U}_k = [\mathbf{X}_k(t_{n+1}) - \mathbf{X}_k(t_n)]/\Delta t$. We calculate the edge-vertex contact volume using Eq. (2.2.8):

$$V_k(\mathbf{e}, \mathbf{X}) = (t - \tau_I)(1 + (\mathbf{U}_k \cdot \mathbf{n}(\tau_I))^2)^{1/2}|\boldsymbol{e}|, \tag{2.3.2}$$

where $\mathbf{n}(\tau_I)$ is the normal to the edge $\mathbf{e}(\tau_I)$. For each edge-vertex contact volume, we calculate the gradient with respect to the vertices $\mathbf{X}_i$, $\mathbf{X}_{i+1}$ and $\mathbf{X}_j$, summing over all the edge-vertex contact pairs we get the total space-time contact volume and gradient.

2.3.2 Temporal discretization

Our temporal discretization is based on the locally-implicit time-stepping scheme in [86] — adapting the Implicit–Explicit (IMEX) scheme [4] for interfacial flow — in which we treat intra-particle interactions implicitly and inter-particle interactions explicitly. We combine this method with the minimal-separation constraint. We refer to this scheme as *constrained locally-implicit* (CLI) scheme. For comparison purposes, we also consider the same scheme without constraints (LI) and the *globally semi-implicit* (GI) scheme, where all interactions treated implicitly [88]. From the perspective of boundary integral formulation, the distinguishing factor between LI and CLI is the extra traction jump term due to collision. Schemes LI/CLI and GI differ in their explicit or implicit treatment of the complementary velocities.

While treating the inter-vesicle interactions explicitly may result in more frequent violations of minimal-separation constraint, we demonstrate that in essentially all cases the CLI scheme is significantly more efficient than both the GI and LI schemes because these schemes are costlier and require higher spatial and temporal resolution to prevent collisions.

We consider two versions of the CLI scheme, a simple first-order Euler scheme and a spectral deferred correction version. A first-order backward Euler CLI time stepping formulation for Eq. (2.2.31) is

$$\frac{1+\nu_i}{2}\mathbf{u}_i^+ = \bar{\mathbf{u}}_i^\gamma + \mathbf{G}_{\gamma_i}\mathbf{f}_i(\mathbf{X}_i^+, \sigma_i^+, \lambda^+) + (1-\nu_i)\mathbf{T}_{\gamma_i}\mathbf{u}_i^+, \tag{2.3.3}$$

$$\mathbf{X}_{i,s}\cdot\mathbf{u}_{i,s}^+ = 0, \tag{2.3.4}$$

$$\mathbf{f}_i(\mathbf{X}_i^+, \sigma_i^+, \lambda^+) = -\kappa_b\mathbf{X}_{i,ssss}^+ - (\sigma^+\mathbf{X}_{i,s})_s + ((\,\mathrm{d}_{\boldsymbol{u}}\mathsf{V}^+)^T\lambda^+)_i, \tag{2.3.5}$$

$$0 \le \mathsf{V}(\gamma; t^+) \quad \perp \quad \lambda^+ \ge 0, \tag{2.3.6}$$

where the implicit unknowns to be solved for at the current step are marked with superscript "+". The position and velocity of the points of i^{th} vesicle are denoted by $\mathbf{X}_i$, $\mathbf{u}_i^+ = (\mathbf{X}_i^+ - \mathbf{X}_i)/\Delta t$, and $\mathbf{f}_i$ is the traction jump on the i^{th} vesicle boundary. $\mathsf{V}(\gamma; t^+)$ is the STIV function.

2.3.2.1 Spectral Deferred Correction

We use spectral deferred correction (SDC) method [62, 83, 84] to get a better stability behavior compared to the basic backward Euler scheme described above. We use SDC both for LI and CLI time-stepping. To obtain the SDC time-stepping equations, we reformulate Eq. (2.2.3) as a Picard integral

$$\boldsymbol{X}(t_{n+1}) = \boldsymbol{X}(t_n) + \int_{t_n}^{t_{n+1}} \boldsymbol{u}\left(\boldsymbol{X}(\tau), \sigma(\tau), \lambda(\tau)\right)\, \mathrm{d}\tau, \tag{2.3.7}$$

where the velocity satisfies Eqs. (2.2.31) and (2.3.3). In the SDC method, the temporal integral in Eq. (2.3.7) is first discretized with $p+1$ Gauss-Lobatto quadrature points [62, 84]. Each iteration starts with *p provisional positions* $\widetilde{\mathbf{X}}$ corresponding to times τ_i in the interval $[t_n, t_{n+1}]$; $t_n = \tau_0 < \cdots < \tau_p = t_{n+1}$. Provisional tensions $\widetilde{\sigma}$ and provisional $\widetilde{\lambda}$ are defined similarly. The SDC method iteratively corrects the provisional positions $\widetilde{\mathbf{X}}$ with the error term $\widetilde{\mathbf{e}}$, which is solved using the residual $\widetilde{\mathbf{r}}$ resulting from the provisional solution as defined below. The residual is given by:

$$\widetilde{\boldsymbol{r}}(\tau) = \widetilde{\boldsymbol{X}}(t_n) - \widetilde{\boldsymbol{X}}(\tau) + \int_{t_n}^{\tau} \widetilde{\boldsymbol{u}}(\theta)\, \mathrm{d}\theta. \tag{2.3.8}$$

After discretization, we use $\widetilde{\mathbf{X}}^{w,m}$, to denote the provisional position at mth Gauss-Lobatto point after w SDC passes. The error term $\widetilde{\mathbf{e}}^{w,m}$ denotes the computed correction to obtain mth provisional position in wth pass. The SDC correction iteration is defined by

$$\widetilde{\mathbf{X}}^{w,m} = \widetilde{\mathbf{X}}^{w-1,m} + \widetilde{\mathbf{e}}^{w,m}, \quad \widetilde{\sigma}^{w,m} = \widetilde{\sigma}^{w-1,m} + \widetilde{\mathbf{e}}_{\sigma}^{w,m}, \quad \widetilde{\lambda}^{w,m} = \widetilde{\lambda}^{w-1,m} + \widetilde{\mathbf{e}}_{\lambda}^{w,m}. \tag{2.3.9}$$

Setting $\widetilde{\mathbf{X}}^{0,m}$ to zero, the first SDC pass is just backward Euler time stepping to obtain nontrivial provisional solutions. Beginning from the second pass, we solve for the error term as corrections.

Denote $(\alpha\mathbf{I} - (1-\nu)\mathbf{T}_{\widetilde{\gamma}^{w-1,m}})$ by $\mathbf{D}_{\widetilde{\gamma}^{w-1,m}}$. Following [83, 84], we solve the following equation for the error term:

$$\alpha\frac{\widetilde{\mathbf{e}}^{w,m}-\widetilde{\mathbf{e}}^{w,m-1}}{\Delta\tau}=\mathbf{D}_{\widetilde{\gamma}^{w-1,m}}\left(\frac{\widetilde{\mathbf{r}}^{w-1,m}-\widetilde{\mathbf{r}}^{w-1,m-1}}{\Delta\tau}\right)+\mathbf{G}_{\widetilde{\gamma}^{w-1,m}}\mathbf{f}(\widetilde{\mathbf{e}}^{w,m},\widetilde{\mathsf{e}}_{\sigma}^{w,m},\widetilde{\mathsf{e}}_{\lambda}^{w,m})+ \\ (1-\nu)\mathbf{T}_{\widetilde{\gamma}^{w-1,m}}\left(\frac{\widetilde{\mathbf{e}}^{w,m}-\widetilde{\mathbf{e}}^{w,m-1}}{\Delta\tau}\right). \tag{2.3.10}$$

Eq. (2.3.10) is the identical to Eq. (2.3.3), except the right-hand-side for Eq. (2.3.10) is obtained from the residual while the right-hand-side for Eq. (2.3.3) is the complementary velocity. The residual $\widetilde{\mathbf{r}}^{w,m}$ is obtained using a discretization of Eq. (2.3.8):

$$\widetilde{\mathbf{r}}^{w,m}=\widetilde{\mathbf{X}}^{w,0}-\widetilde{\mathbf{X}}^{w,m}+\sum_{l=0}^{p}w_{l,m}\widetilde{\mathbf{u}}^{w,l}. \tag{2.3.11}$$

where $w_{l,m}$ are the quadrature weights for Gauss-Lobatto points, whose quadrature error is $\mathcal{O}(\Delta t^{2p-3})$. In addition to the SDC iteration, Eq. (2.3.10), we also enforce the inextensibility constraint

$$\widetilde{\mathbf{X}}_{s}^{w-1,m}\cdot\widetilde{\mathbf{u}}_{s}^{w,m}=0, \tag{2.3.12}$$

and the contact complementarity

$$0\le\mathsf{V}(\widetilde{\gamma}^{w,m})\quad\perp\quad\widetilde{\mathsf{e}}_{\lambda}^{w,m}\ge 0. \tag{2.3.13}$$

In evaluating the residuals using Eq. (2.3.11), provisional velocities are required. In the GI scheme [84], all the interactions are treated implicitly and given provisional position $\widetilde{\mathbf{X}}^{w,m}$, the provisional velocities are obtained by evaluating

$$\widetilde{\mathbf{u}}^{w,m}=\mathbf{D}_{\widetilde{\gamma}^{w,m}}^{-1}\left(\mathbf{G}_{\widetilde{\gamma}^{w,m}}\mathbf{f}(\widetilde{\mathbf{X}}^{w,m},\widetilde{\sigma}^{w,m},\widetilde{\lambda}^{w,m})+\mathbf{u}^{\infty}\right). \tag{2.3.14}$$

which requires a global inversion of $\mathbf{D}_{\widetilde{\gamma}^{w,m}}$. The same approach is taken for LI and CLI schemes, except the provisional velocities are obtained using local inversion only, all the inter-particle interactions are treated explicitly and added to the explicit term, i.e., complementary velocity $\widetilde{\widetilde{\mathbf{u}}}_i^{w-1,m}$; modifying Eq. (2.3.14) for each vesicle, we obtain

$$\widetilde{\mathbf{u}}_i^{w,m} = \mathbf{D}^{-1}_{\widetilde{\gamma}_i^{w,m}} \left(\mathbf{G}_{\widetilde{\gamma}_i^{w,m}} \mathbf{f}(\widetilde{\mathbf{X}}_i^{w,m}, \widetilde{\sigma}_i^{w,m}, \widetilde{\lambda}_i^{w,m}) + \widetilde{\widetilde{\mathbf{u}}}_i^{w-1,m} \right), \tag{2.3.15}$$

where $\widetilde{\widetilde{\mathbf{u}}}_i^{w-1,m}$ is computed using $\widetilde{\mathbf{X}}^{w,m}$ and $\widetilde{\mathbf{u}}_j^{w-1,m}(j \neq i)$ accounting the velocity influence from other vesicles. We only need to invert the local interaction matrices $\mathbf{D}_{\widetilde{\gamma}_i^{w,m}}$ in this scheme.

2.3.2.2 Contact-resolving iteration

Let $\mathbf{AX}^+ = \mathbf{b}$ be the linear system that is solved at each iteration of a CLI scheme (in case of the CLI-SDC scheme, on each of the inner step of the SDC). $\mathbf{A}$ is a block diagonal matrix, with blocks $\mathbf{A}_{ii}$ corresponding to the self interactions of the i^{th} particle. All inter-particle interactions are treated explicitly, and thus included in the right-hand side $\mathbf{b}$. We write Eq. (2.3.3), or Eq. (2.3.10), in a compact form as

$$\mathbf{AX}^+ = \mathbf{b} + \mathbf{Gf}_c^+, \tag{2.3.16}$$

$$0 \leq \mathsf{V}(\gamma; t^+) \quad \perp \quad \lambda \geq 0, \tag{2.3.17}$$

which is a mixed Nonlinear Complementarity Problem (NCP), because the STIV function $\mathsf{V}(\gamma, t)$ is a nonlinear function of position. Since this is the CLI scheme, $\mathbf{G}$ is a block diagonal matrix. To approximately solve this NCP, we use a first-order linearization of the $\mathsf{V}(\gamma;t)$ to obtain an LCP and iterate until the NCP is solved to the desired accuracy:

$$\mathbf{AX}^\star = \mathbf{b} + \mathbf{GJ}^T\lambda. \tag{2.3.18}$$

$$0 \le \mathsf{V}(\gamma;t^{+k}) + \mathbf{J}\Delta\mathbf{X} \quad \perp \quad \lambda \ge 0, \tag{2.3.19}$$

where $\Delta\mathbf{X}$ is the update to get the new candidate solution $\mathbf{X}^\star$, and $\mathbf{J}$ denotes the Jacobian of the volume $\nabla_X \mathsf{V}(\gamma, t^{+k})$.

Algorithm 1 summarizes the steps to solve Eqs. (2.3.16) and (2.3.17) as a series of linearization steps Eqs. (2.3.18) and (2.3.19). We discuss the details of the LCP solver separately below. In lines 1 to 6, we solve the unconstrained system $\mathbf{AX}^\star = \mathbf{b}$ using the solution from previous time step. Then, the STIVs are computed to check for collision. The loop in lines 7−14 is the linearized contact-resolving steps.

Algorithm 1: CONTACT-FREE TIME-STEPPING.

```
   input : $\mathbf{X}, \mathbf{b}$
   output: $\mathbf{X}^+, \mathbf{f}_c^+$
 1 $\mathbf{A} \leftarrow \mathbf{A}(\mathbf{X})$
 2 $\mathbf{b} \leftarrow \mathbf{b}(\mathbf{X})$
 3 $\mathbf{f}_c^+ \leftarrow 0$
 4 $k \leftarrow 0$
 5 $\mathbf{X}^\star \leftarrow \mathbf{A}^{-1}\mathbf{b}$
 6 $\mathsf{V} \leftarrow$ getContactVolume($\mathbf{X}^\star$)
 7 while $\mathsf{V} < 0$ do
 8     $\mathbf{J} \leftarrow$ getContactVolumeJacobi($\mathbf{X}^\star$)
 9     $\lambda \leftarrow$ lcpSolver(V)
10     $k \leftarrow k + 1$
11     $\mathbf{b} \leftarrow \mathbf{b} + \mathbf{GJ}^T\lambda$
12     $\mathbf{X}^\star \leftarrow \mathbf{A}^{-1}\mathbf{b}$
13     $\mathsf{V} \leftarrow$ getContactVolume($\mathbf{X}^\star$)
14     $\mathbf{f}_c^+ \leftarrow \mathbf{f}_c^+ + \mathbf{J}^T\lambda$
15 $\mathbf{X}^+ \leftarrow \mathbf{X}^\star$
```

Substituting Eq. (2.3.18) into Eq. (2.3.19), and using the fact that $\Delta\mathbf{X} = \mathbf{A}^{-1}\mathbf{G}\mathbf{J}^T\lambda$ we cast the problem in the standard LCP form

$$0 \leq \mathsf{V} + \mathbf{B}\lambda \quad \perp \quad \lambda \geq 0, \tag{2.3.20}$$

where $\mathbf{B} = \mathbf{J}\mathbf{A}^{-1}\mathbf{G}\mathbf{J}^T$. The LCP solver is called on line 9 to obtain the magnitude of the constraint force, which is in turn used to obtain new candidate positions that may or may not satisfy the constraints. In line 11, the collision force is incorporated into the right-hand-side $\mathbf{b}$ for self interaction in the next LCP iteration. Line 13 checks the minimal-separation constraints for the candidate solution. In line 14, the contact force is updated, which will be used to form the right-hand-side $\mathbf{b}$ for the global interaction in the next time step.

The LCP matrix $\mathbf{B}$ is an M by M matrix, where M is the number of contact volumes, $M = \mathcal{O}(N_v + N_p)$. Each entry $\mathbf{B}_{k,p}$ is the induced change in the k^{th} contact volume by the p^{th} contact force. Matrix $\mathbf{B}$ is sparse and typically diagonally dominant, since most STIV volumes are spatially separate.

2.3.3 Solving the Linear Complementarity Problem

In the contact-resolving iterations we solve an LCP, Eq. (2.3.20). Most common algorithms (e.g., Lemke's algorithm [48] and splitting based iterative algorithms [1, 60]) requires explicitly formed LCP matrix $\mathbf{B}$, which can be prohibitively expensive when there are many collisions. We use the minimum-map Newton method [15, Section 5.8], which we modify to require matrix-vector evaluation only, as we can perform it without explicitly forming the system matrix.

We briefly summarize the minimum-map Newton method. Let $\mathbf{y} = \mathsf{V} + \mathbf{B}\lambda$.

Using the minimum map reformulation we can convert the LCP to a root-finding problem

$$\mathbf{H}(\lambda) \equiv \begin{bmatrix} h(\lambda_1, y_1) \\ \dots \\ h(\lambda_M, y_M) \end{bmatrix} = 0, \tag{2.3.21}$$

where $h(\lambda_i, y_i) = \min(\lambda_i, y_i)$. This problem is solved by Newton's method (Alg. 2). In the algorithm, $\mathbf{P}_{\mathbb{A}}$ and $\mathbf{P}_{\mathbb{F}}$ are selection matrices: $\mathbf{P}_{\mathbb{A}}\lambda$ selects the rows of λ whose indices are in set $\mathbb{A}$ and zeros out all the other rows. While function $\mathbf{H}$ is not smooth, it is Lipschitz and directionally differentiable, and its so-called B-derivative $\mathbf{P}_{\mathbb{A}}\mathbf{B} + \mathbf{P}_{\mathbb{F}}$ can be formed to find the descent direction for Newton's method [21]. The matrix $\mathbf{P}_{\mathbb{A}}\mathbf{B} + \mathbf{P}_{\mathbb{F}}$ is a sparse matrix, and we use GMRES to solve this linear system. Since $\mathbf{B}$ is sparse and diagonally dominant, in practice the linear system is solved in few GMRES iterations and the Newton solver converges quadratically.

Algorithm 2: MINIMUM MAP LCP SOLVER.

```
   require: applyLCPMatrix(), $\mathsf{V}$ and $\epsilon$
   output : $\lambda$
1  $e \leftarrow \epsilon$
2  $\lambda \leftarrow 0$
3  while $e > \epsilon$ do
4      $\mathbf{y} \leftarrow \mathsf{V} +$ applyLCPMatrix($\lambda$)
5      $\mathbb{A} \leftarrow \{i | y_i < \lambda_i\}$                 // index of active constraints
6      $\mathbb{F} \leftarrow \{i | y_i \geq \lambda_i\}$
7      Iteratively solve $\begin{bmatrix} \mathbf{B} & -\mathbf{I} \\ \mathbf{P}_{\mathbb{F}} & \mathbf{P}_{\mathbb{A}} \end{bmatrix} \begin{bmatrix} \Delta\lambda \\ \Delta y \end{bmatrix} = \begin{bmatrix} 0 \\ -\mathbf{P}_{\mathbb{A}}\mathbf{y} - \mathbf{P}_{\mathbb{F}}\lambda \end{bmatrix}$     // $\mathbf{B}$ applied
        by applyLCPMatrix
8      $\tau \leftarrow$ projectLineSearch($\Delta\lambda$)
9      $\lambda \leftarrow \lambda + \tau\Delta\lambda$
10     $e \leftarrow \|\mathbf{H}(\lambda)\|$
```

2.3.4 Algorithm Complexity

We estimate the complexity of a single time step as a function of the number of points on each vesicle N, number of vesicles N_v. Let C_N denote the cost of solving a local linear system for one particle; then the complexity of inverting linear systems for all particles is $\mathcal{O}\left(C_N N_v\right)$. In [86, 103] it is shown that for LI scheme $C_N = \mathcal{O}\left(N \log N\right)$. The cost of evaluating the inter-particle interactions at the $N_v N$ discrete points using FMM is $\mathcal{O}(N_v N)$.

We assume that for each contact resolving step, the number of contact volumes is M. Assuming that minimum map Newton method takes K_1 steps to converge, the cost of solving the LCP is $\mathcal{O}\left(K_1 C_N N_v\right)$, because inverting A is the costliest step in applying the LCP matrix. The total cost of solving the NCP problem is $\mathcal{O}\left(K_1 K_2 C_N N_v\right)$, where K_2 are the number of contact resolving iterations. In the numerical simulations we observe that the minimum map Newton method converges in a few iterations ($K_1 \approx 15$) and the number of contact resolving iterations is also small and independent of the problem size ($K_2 \approx 10$). In Section 2.4, we compare the cost of solving contact constrained system and the cost of unconstrained system.

2.4 Results

In this section, we present results characterizing the accuracy, robustness, and efficiency of a locally-implicit time stepping scheme (CLI) combined with our contact resolution framework in comparison to other schemes mention in Section 2.3.2 with no contact resolution (i.e., LI and GI schemes).

- First, to demonstrate the robustness of our scheme in maintaining the prescribed minimal separation distance with different viscosity contrast ν, we consider two vesicles in an extensional flow, Section 2.4.1.

- In Section 2.4.2, we explore the effect of minimal separation d_m and its effect on collision displacement in shear flow. We demonstrate that the collision scheme has a minimal effect on the shear displacement.
- We compare the cost of our scheme with the unconstrained system using a simple sedimentation example in Section 2.4.3. While the per-step cost of the unconstrained locally implicit system is marginally lower, it requires much finer spatial and temporal resolutions in order to maintain a valid contact-free configuration, making the overall cost prohibitive.
- We report the convergence behavior of different time-stepping in Section 2.4.4 and show that our scheme achieves second order convergence rate with SDC2.
- We illustrate the efficiency and robustness of our algorithm with three examples: 100 sedimenting vesicles in a container, 196 vesicles in the Couette apparatus with 48% volume fraction (Section 2.4.5), and a flow with multiple vesicles and rigid particles within a constricted tube in Section 2.4.6.

Our experiments support the general observation that when vesicles become close, the LI scheme does a very poor job in handling of vesicles' interaction [88] and the time stepping becomes unstable. The GI scheme stays stable, but the iterative solver requires more and more iterations to reach the desired tolerance, which in turn implies higher computational cost for each time step. Therefore, the GI scheme performance degrades to the point of not being feasible due to the cost of computation and the LI scheme fails due to intersection or the time-step

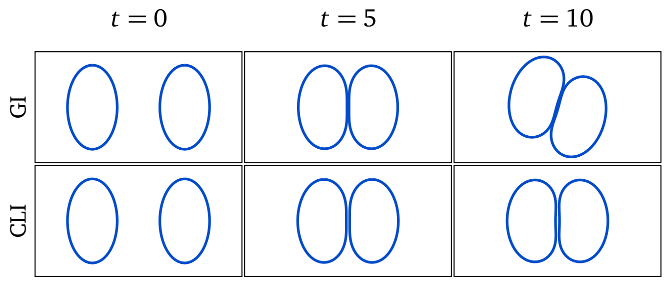


Figure 2.2: SNAPSHOTS OF TWO VESICLES IN EXTENSIONAL FLOW USING GI AND CLI SCHEMES. *As the distance between two vesicles decreases the configuration loses symmetry in the GI scheme as shown in the top row. Nevertheless, as shown in the second row, the LI with minimal-separation constraint scheme maintains the desired minimum separation distance and two vesicles also maintain a symmetric configuration. (The viscosity contrast is 500 in this simulation).*

instability.

2.4.1 Extensional flow

To demonstrate the robustness of our collision resolution framework, we consider two vesicles placed symmetrically with respect to the y axis in the extensional flow $\boldsymbol{u} = [-x, y]$. The vesicles have reduced area of 0.9 and we use a first-order time stepping with LI, CLI, and GI schemes for the experiments in this test. We run the experiments with different time step size and viscosity contrast and report the minimal distance between vesicles as well as the final error in vesicle perimeter, which should be kept constant due local inextensibility. Snapshots of the vesicle configuration for two of the time-stepping schemes are shown in Fig. 2.2.

In Fig. 2.3a, we plot the distance between two vesicles over time. The vesicles continue to get closer in the GI scheme. However, the CLI scheme maintains the desired minimum separation distance between two vesicles. In Fig. 2.3b, we show the minimum distance between the vesicles over the course of simulation (with time horizon $T = 10$) versus the viscosity contrast. As expected, we observe that

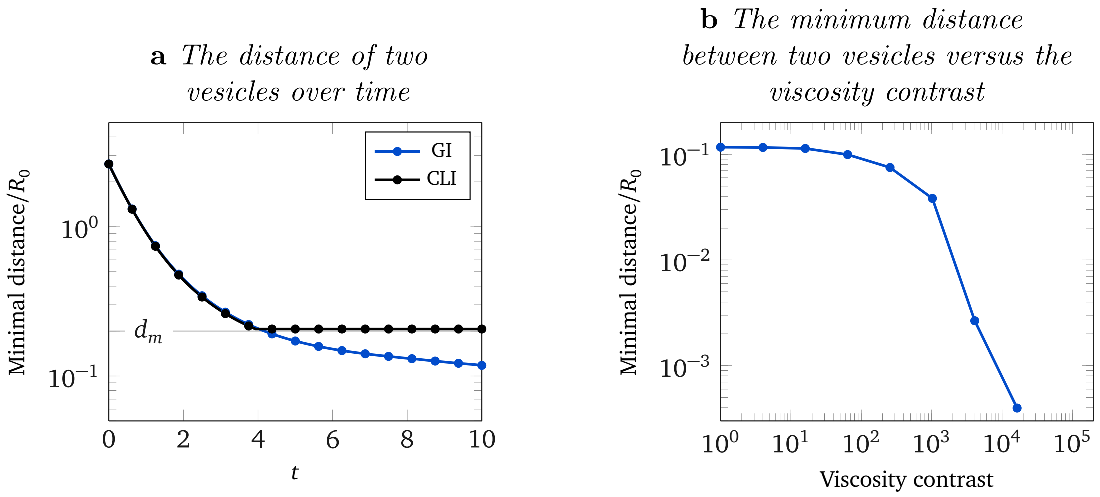


Figure 2.3: DISTANCE BETWEEN TWO VESICLES IN EXTENSIONAL FLOW. *Fig. 2.3a The distance between two vesicles over time for both CLI and GI schemes. $R_0 := L/2\pi$ denotes the effective radius of a vesicle. The CLI scheme easily maintains the prescribed minimal separation of d_m. Fig. 2.3b The final distance (at $T = 10$) between two vesicles as viscosity contrast is increased (using GI scheme).*

the minimum distance between two vesicles decreases as the viscosity contrast is increased. Consequently, for higher viscosity contrast with both GI and LI schemes, either the configuration loses its symmetry or the two vesicles intersect. With minimal-separation constraint, any desired minimum separation distance between vesicles is maintained, and the simulation is more robust and accurate as shown in Fig. 2.2 and Table 2.2.

In Table 2.2, we report the final error in vesicle perimeter for different schemes with respect to viscosity contrast and timestep size. With minimal-separation constraint, we achieve similar or smaller error in length compared to LI or GI methods (when these methods produce a valid result). Moreover, whereas one can use relatively large time step in CLI for all flow parameters — most notably when vesicles have high viscosity contrast — the LI scheme requires very small, often impractical, time steps to prevent instability or intersection.

ν	Δt	CLI	LI	GI
1	0.4	1.17e−01	1.17e−01	7.66e−02
1	0.2	9.49e−04	9.49e−04	1.03e−03
1	0.1	4.49e−04	4.49e−04	4.69e−04
1	0.05	2.23e−04	2.23e−04	2.29e−04
1	0.025	1.12e−04	1.12e−04	1.14e−04
1	0.0125	5.65e−05	5.65e−05	5.68e−05
1e2	0.4	9.42e−03	–	2.54e−04
1e2	0.2	1.33e−04	1.33e−04	1.22e−04
1e2	0.1	6.38e−05	6.38e−05	5.96e−05
1e2	0.05	3.05e−05	3.05e−05	2.95e−05
1e2	0.025	1.49e−05	1.49e−05	1.47e−05
1e2	0.0125	7.39e−06	7.39e−05	7.32e−06
1e3	0.4	8.88e−03$^{T=8.8}$	–	1.66e−05
1e3	0.2	2.08e−02	–	7.99e−06
1e3	0.1	5.85e−06	–	3.93e−06
1e3	0.05	2.42e−06	1.93e−06	1.95e−06
1e3	0.025	1.12e−06	9.78e−07	7.33e−07
1e3	0.0125	5.89e−07	4.87e−07	2.01e−07
1e4	0.4	1.45e−03$^{T=3.6}$	–	–
1e4	0.2	7.23e−04$^{T=5.4}$	–	–
1e4	0.1	7.75e−04$^{T=6.8}$	–	–
1e4	0.05	2.41e−06	–	–
1e4	0.025	1.17e−06	–	–
1e4	0.0125	5.58e−07	–	–

Table 2.2: ERROR IN THE LENGTH OF VESICLES IN EXTENSIONAL FLOW. *The error in the final length of two vesicles in extensional flow with respect to viscosity contrast, timestep size, and for different schemes. The experiment's setup is described in Section 2.4.1 and snapshots of which are shown in Fig. 2.2. The cases with a "–" indicate that either vesicles have intersected or the GMRES failed to converge due to ill-conditioning of the system; the latter happens in the GI scheme and high viscosity contrast. Cases with superscript indicate that the flow loses its symmetry at that time.*

2.4.2 Shear flow

We consider vesicles and rigid bodies in an unbounded shear flow and explore the effects of minimal separation on shear diffusivity. In the first simulation, we consider two vesicles of reduced area 0.98 (to minimize the effect of vesicles' relative orientation on the dynamics) placed in a shear flow with (non-dimensional) shear rate $\chi = 2$. We observed in our previous work that in semi-implicit methods for vesicle suspensions, the stable time step is inversely proportional to shear rate χ [86, Table 6] and [88, Table 4]. This is expected because we use the bending relaxation time as characteristic time (see Section 2.2.5) that becomes less dominant as the shear rate increases. With the observation that $\Delta t^{\text{stable}} \propto \chi^{-1}$, we report the results for a single shear rate, and the approximate stable time step can be estimated for other rates from this.

Let $\delta_t = |y_t^1 - y_t^2|$ denote the vertical offset between the centroids of vesicles at time t. Initially, two vesicles are placed with a relative vertical offset δ_0 as show in Fig. 2.4.

In Fig. 2.5, we report δ_t and its value upon termination of the simulation when $x^1 > 8$, denoted by $\delta_\infty(d_m)$. In Figs. 2.5a and 2.5b, we plot δ_t with respect to x^1 for different minimum separation distances. Based on a high-resolution simulation (with $N = 128$ and $8\times$ smaller time step), the minimal distance between two vesicles without contact constraint is about $2.9h$ for vesicles and $2.2h$ for rigid particles. As the minimum separation parameter d_m is decreased below this threshold, the simulations with minimal-separation constraint converge to the reference

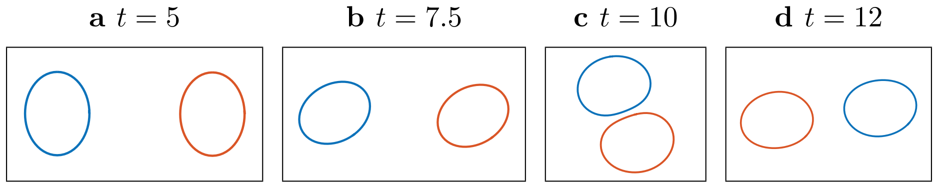


Figure 2.4: SHEAR FLOW EXPERIMENT. *The snapshots of two vesicles in shear flow. Initially, one vesicle is placed at* $[-8, \delta_0]$ *and the second vesicle is placed at* $[0, 0]$.

simulation without minimal-separation constraint. In Fig. 2.5c, we plot the excess terminal displacement due to contact constraint, $[\delta_\infty(d_m) - \delta_\infty(0)]/h$, as a function of the minimum separation distance. When collision constraint is activate, the particles are in effect hydrodynamically larger and the excess displacement grows linearly with d_m.

2.4.3 Sedimentation

To compare the performance of schemes with and without contact constraints, we first consider a small problem with three vesicles sedimenting in a container. We compare three first-order time stepping schemes: locally implicit (LI), locally implicit with collision handling (CLI), and globally implicit (GI).

Snapshots of these simulations are shown in Fig. 2.6. For the LI scheme, the

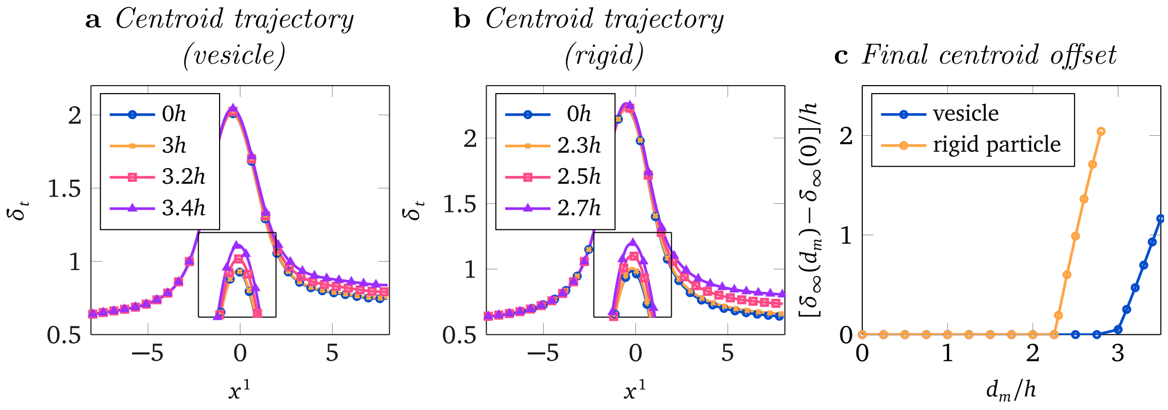


Figure 2.5: The offset $\delta_t(d_m)$ between the centroids of two vesicles in shear flow. *The initial offset is* $\delta_0 = 0.64$ *and* $N = 64$ *discretization points are used, implying* $h = 0.0994$*, where h is the distance between two discretization points along vesicle boundary. Fig. 2.5a and Fig. 2.5b show* $\delta_t(x^1)$ *for the vesicles and circular rigid particles for different minimal separation distance* d_m*. Fig. 2.5c shows the excess displacement induced by minimal separation. When collision constraint is activate (larger* d_m*), the particles are in effect hydrodynamically larger and the excess displacement grows linearly with* d_m*.*

error grows rapidly when the vesicles intersect and a 64× smaller time step is required for resolving the contact and for stability. Similarly, for the GI scheme the vesicle intersect as shown in Figs. 2.6e and 2.6f and a 4× smaller time step is needed for the contact to be resolved. The CLI scheme, maintains the desired minimal separation between vesicles.

Although the current code is not optimized for computational efficiency, it is instructive to consider the relative amount of time spent for a single time step in each scheme. Each time step in LI scheme takes about 1 second, the time goes up to 1.5 seconds for the CLI scheme. In contrast, a single time step with the GI scheme takes, on average, 65 seconds because the solver needs up to 240 GMRES iterations to converge when vesicles are very close. This excessive cost renders this scheme impractical for large problems.

To demonstrate the capabilities of the CLI scheme and to gain a qualitative

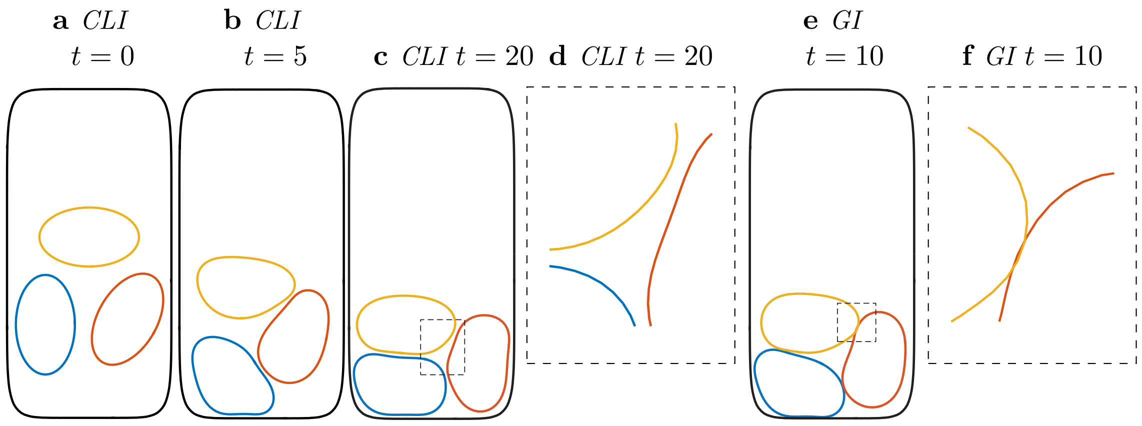


Figure 2.6: SEDIMENTING VESICLES. *Snapshots of three sedimenting vesicles in a container using three time-stepping schemes. The vesicles have reduced area of* 0.9*, their viscosity contrast is* 100*, and are discretized with* 64 *points. The boundary is discretized with* 256 *points, the simulation time horizon* $T = 26$*, and the time step is* 0.01*. Fig. 2.6a–Fig. 2.6c The dynamics of three vesicles sedimenting with CLI scheme. Fig. 2.6d The contact region of Fig. 2.6c. Fig. 2.6e The instance when the vesicles intersect using GI scheme. Fig. 2.6f The contact region of Fig. 2.6e.*

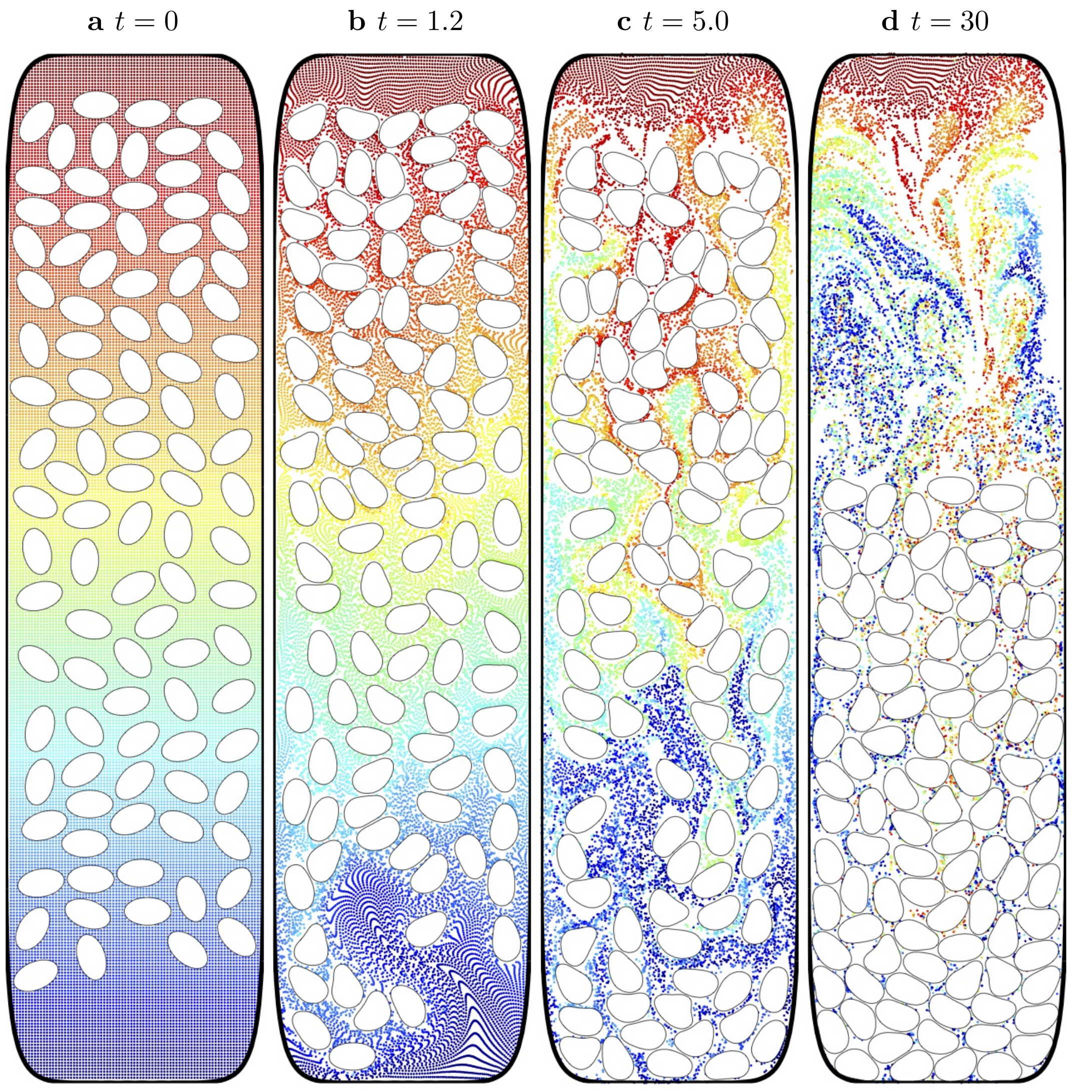


Figure 2.7: SEDIMENTATION OF ONE HUNDRED VESICLES. *Vesicles are randomly placed in a container, where only gravitational force is present. Fig. 2.7a The initial configuration. The colored tracer particles are shown for visualization purposes and are not hydrodynamically active. Fig. 2.7b and Fig. 2.7c The intermediate states; we see that as vesicles move downward they induce a strong back flow. Fig. 2.7d shows the state where the simulation was terminated. As the vesicles accumulate at the bottom of the container, many collision areas stay active. Nonetheless, the vesicles are stacked stably and without any artifact from the collision handling.*

understanding of the scaling of the cost as the number of intersections increases, we consider the sedimentation of 100 vesicles. As the sediment progresses the number of contact regions grows to about 70 per time step. For this simulation, we use a lower viscosity contrast of 10 and the enclosing boundary is discretized with 512 points and the total time is $T = 30$. Snapshots of the simulation with CLI scheme are shown in Fig. 2.7.

We observe that with first-order time stepping, both LI and CLI schemes are unstable. Therefore, we run this simulation with second order SDC using LI and CLI schemes. The GI scheme is prohibitively expensive and infeasible in this case, due to ill-conditioning and large number of GMRES iterations per time-step.

The LI scheme requires at least 12000 time steps to maintain the non-intersection constraint and each time step takes about 700 seconds to complete. On the other hand, the CLI scheme only need 1500 time steps to complete the simulation (taking the stable time step) and each time step takes about 830 seconds. We repeated this experiment with 16 discretization points on each vesicle using CLI scheme and 1500 time steps are also sufficient for this case (with final length error about $3.83e{-3}$), each time step takes about 170 seconds and the simulation takes about 70 hours to complete. The number of contact resolving iterations in Alg. 1 is about 10.

In summary, LI scheme requires $4\times$ more points on each vesicle and $8\times$ smaller time-step size to keep vesicles in a valid configuration compared to CLI scheme.

2.4.4 Convergence analysis

To investigate the accuracy of the time-stepping schemes, we consider the sedimentation of three vesicles with reduced area 0.9 in a container as shown in Fig. 2.6. We test LI and CLI schemes for a range of time steps and spatial resolutions and report the error in the location of the center of mass of the vesicles at the end of simulation. The spatial resolution h is chosen proportional to time-step size and for the CLI scheme the minimal separation d_m is proportional to h. As a reference, we use a fine-resolution simulation with GI scheme. The error as a function of time step size is shown in Fig. 2.8.

2.4.5 Couette Apparatus

To demonstrate the ability of the contact constraint scheme to handle a high volume fraction of vesicles, we consider the flow inside a Couette apparatus. The device is filled with 196 vesicles of reduced area 0.65 and viscosity contrast 2. The volume fraction is approximately 48%. With this high concentration, we use $\Delta t = 0.04$ and SDC2 for both LI and CLI schemes. Snapshots of the flow and the instantaneous effective viscosity are illustrated in Fig. 2.9.

The LI scheme results in intersecting vesicles at $t = 10.6$, while the CLI scheme maintains the contact constraint over the whole simulation. There are approximately 10 contact regions per time step. The simulation with $T = 20$ takes about

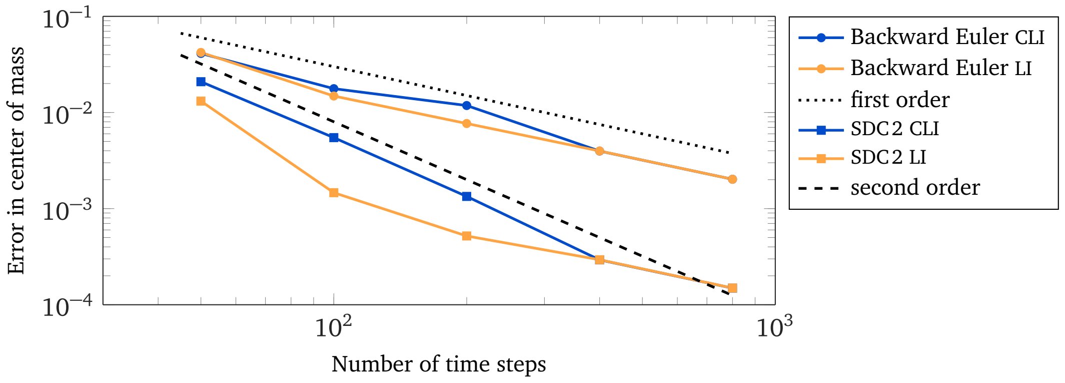


Figure 2.8: CONVERGENCE RATE. *We compare the final center of mass of three sedimenting vesicles in a container (shown in Fig. 2.6) for Backward Euler and SDC2 (second order Spectral Deferred Correction scheme). The time horizon is set to $T = 2$. We choose the spatial resolution proportional to the time-step. As a reference, we use the results for the GI scheme with time step $\Delta t = 1.25e{-3}$ and $N = 256$ discretization points on each vesicle and 512 discretization points on the boundary.*

120 hours to complete for CLI scheme. At $t = T$, the error in area is $7.81e-03$ and the error in length is $1.17e-03$. We did not run the GI scheme as, with the same area and length errors, it is estimated to take more than three months for the simulation to complete.

In Fig. 2.9c we report the instantaneous effective viscosity with respect to time. The effective viscosity is in qualitative agreement with similar studies for semi-dilute rigid-particle suspensions in wall bounded shear flow [16, Eq. 1 with $[\eta] = 2, \beta = 3.6$]. We are not aware of any other study for the effective viscosity of high-volume fraction vesicle suspension with flow curvature and viscosity contrast. An expression for effective viscosity as a function of the flow parameters can be constructed from this type of experiment by systematic long-time integration of multiple instantiations of the flow with large ensembles which we will report in a separate work.

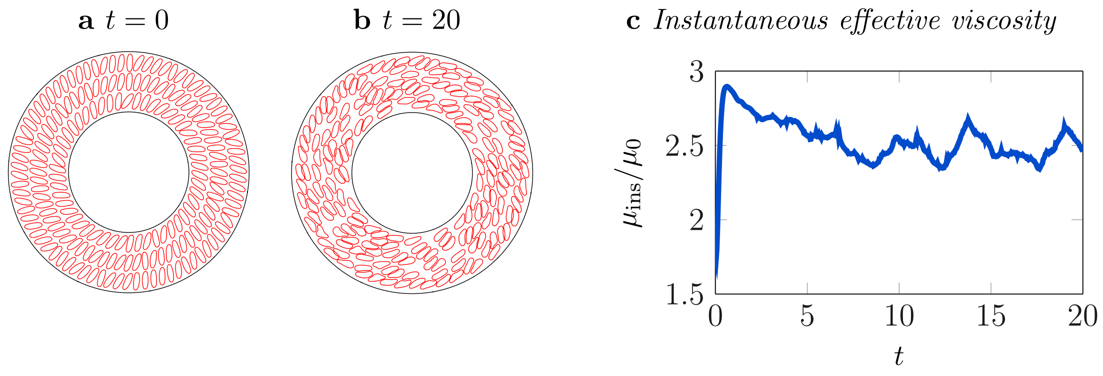


Figure 2.9: COUETTE FLOW WITH 196 VESICLES. *Fig. 2.9a–Fig. 2.9b Snapshots of the vesicles' initial and final configuration with the contact constraint. Vesicles a viscosity contrast of* 2 *and their reduced area is* 0.65*. The inner boundary completes one full revolution every 10 time units. The simulation without contact constraint fails at* $t = 10.64$ *as vesicles intersect. Fig. 2.9c The instantaneous effective viscosity (as the ratio of the net torque on the inner circle and its angular velocity) with respect to time.*

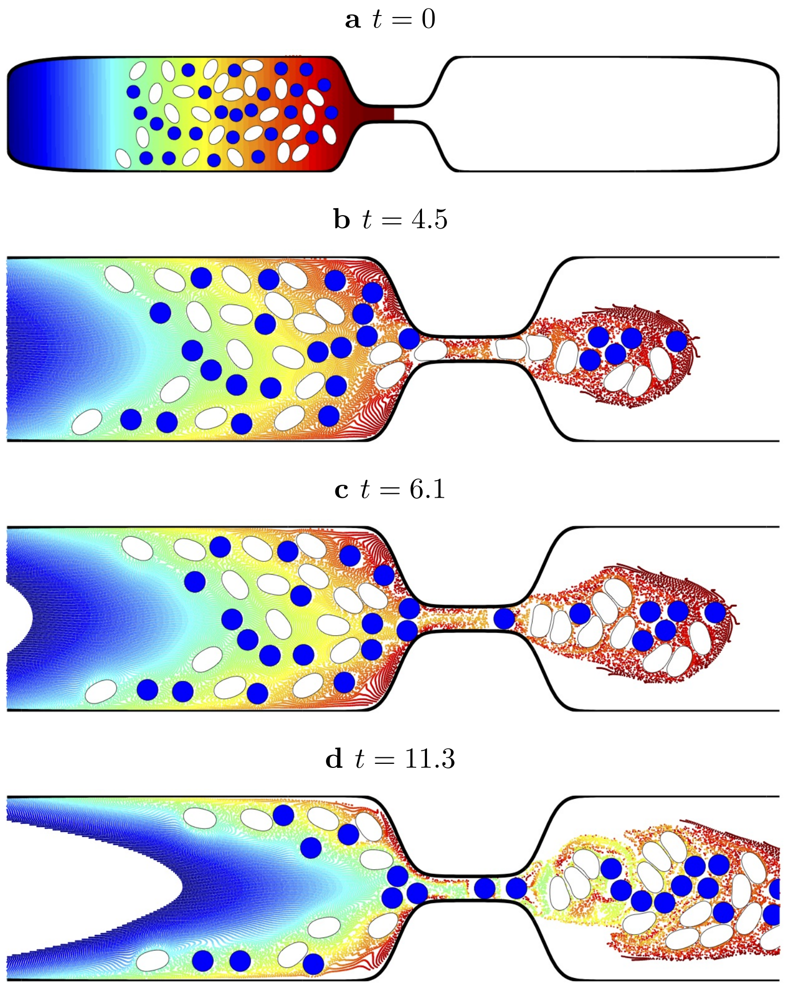


Figure 2.10: Stenosis with 25 vesicles and 25 rigid particles. *Snapshots of vesicles and rigid particles passing a constricted tube, the fluid flows from left to right (using proper Dirichlet boundary condition on the wall). The colored tracers are for visualization purposes and do not induce any flow. The simulation time horizon is set to* $T = 20$*, each vesicle or rigid particle is discretized with* 32 *points, and the wall is discretized with* 1024 *points. Fig. 2.10a The initial configuration. Fig. 2.10b–Fig. 2.10d The interaction and collision between vesicles, rigid particles, and the domain boundary at different instances. Without minimal-separation constraint, vesicles and rigid particles easily contact at the entrance of the constricted area.*

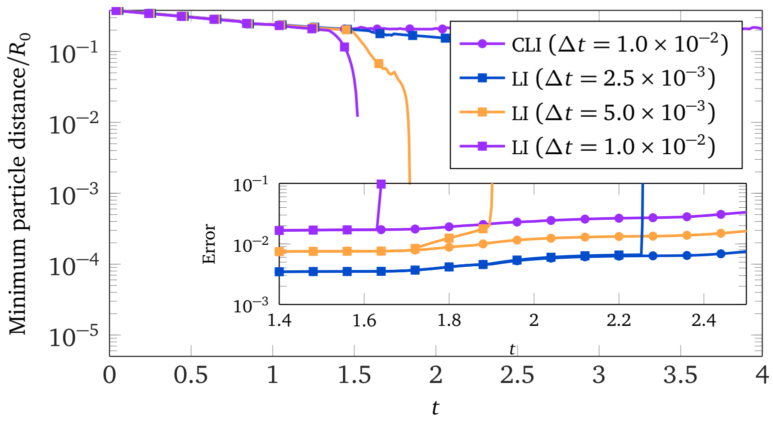


Figure 2.11: ERROR FOR STENOSIS EXAMPLE. *The relative minimal distance and error for different schemes and time step sizes.* R_0 *is the effective radius. For the LI scheme, independent of the time step size, vesicles or particles intersect with the domain boundary, resulting in exponential error growth. The CLI scheme is stable and maintains the desired minimal distance.*

2.4.6 Stenosis

As another stress test for particle-boundary and vesicle-boundary interaction, we study the flow with 25 vesicles of reduced area 0.9, mixed with 25 circular rigid particles in a constricted tube (Fig. 2.10). It is well known that rigid bodies can become arbitrarily close in the Stokes flow, e.g. [51], and without proper collision handling, the required temporal/spatial resolution is expected to be high for methods based on boundary integrals; [85] and Fig. 2.11.

In this example, the vesicles and rigid particles are placed at the right hand side of the constricted tube. We use backward Euler method and search for the stable time step for schemes LI and CLI. Similarly to the sedimentation example, we do not consider the GI scheme due to its excessive computational cost.

The stable time step for the CLI scheme is $\Delta t = 0.01$. For the LI scheme, we tested the cases with up to 32× smaller time step size, but we were unable to avoid contact and intersection between vesicles and rigid particles.

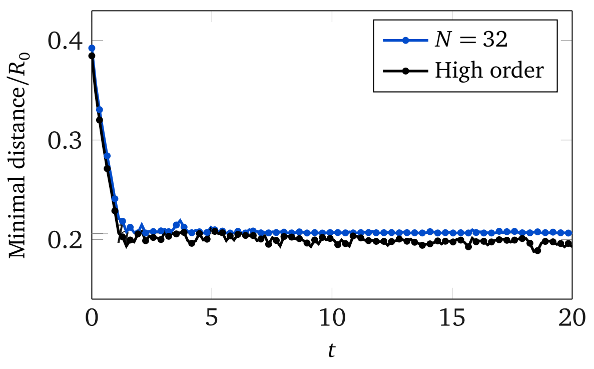


Figure 2.12: Effective minimal distance. *The minimal distance between 32-point piecewise-linear approximations (enforced by the constraint) and the true minimal distance between high-order shapes. R_0 is the effective radius of vesicles. We use an upsampled linear approximation with $N = 128$ as the surrogate for the high-order curves.*

Figure 2.11 shows the error and minimal distance between vesicles, particles, and boundary for CLI and LI schemes with different time step sizes. Without the minimal-separation constraint, the solution diverges when the particles cross the domain boundary.

To validate our estimates for the errors due to using a piecewise-linear approximation in the minimal separation constraint instead of high-order shapes, we plot the minimum distance at each step for the piecewise-linear approximation and upsampled shapes in Fig. 2.12. The target minimal separation distance is set to $d_m = h$. We observe that the actual minimal distance for smooth curve is smaller than the minimal distance for piecewise-linear curve, while the difference between two distances is small compared to the target minimum separation distance.

Note that the due to higher shear rate in the constricted area, the stable time step is dictated by the dynamics in that area. For these flows, we expect a combination of adaptive time stepping [84] and the scheme outlined in this chapter to provide the highest speedup.

2.5 Conclusion

In this chapter we introduced a new scheme for efficient simulation of dense suspensions of deformable and rigid particles immersed in Stokesian fluid. We demonstrated through numerical experiments that this scheme is orders of magnitude faster than the alternatives and can achieve high order temporal accuracy.

Chapter 3

Parallel contact-aware simulations of deformable particles in three-dimensional free-space Stokes flow

3.1 Introduction

Suspensions of rigid and deformable particles are ubiquitous in biological systems and industrial applications. Well-known examples of such fluids are colloids, particulate suspensions, soft-particle pastes, cytoplasm, and blood. The rheology of some these suspensions is well understood in the dilute regime where theoretical models are tractable and simplifying assumptions can be applied to both theoretical and computational models.

In these suspensions, as the volume fraction of particles increases, collective motion emerges and the length scale of the problem grows [49], whereby making theoretical analysis intractable and high-fidelity computational models fraught with numerical challenges. A critical component in simulating *non-dilute* suspensions of (deformable) particles is robust and accurate collision handling. From the rheological perspective, proper collision handling is key for obtaining accurate bulk properties as well as observing correct phase transitions (e.g., to clogging). Computationally, robust collision handling algorithms are key for stable long-time simulations of large ensembles.

To this end, we present an efficient, accurate, and robust method for parallel simulation of dense suspensions in Stokesian fluid in 3D (e.g., Fig. 3.8). This work complements our previous work [53] by extending it to 3D and by introducing new parallel algorithms for collision detection and handling, essential for modeling large numbers of particles. We report computational experiments with volume fractions up to 60% (above the volume fraction of red blood cells, i.e., 45%) involving thousands of deformable particles. We also report strong and weak scaling results for the parallel algorithms on the Stampede system at Texas Advanced Computing Center.

Similar to [53], we focus on the Stokes flow and the *vesicle* model for deformable particles. Vesicles are closed deformable membranes suspended in a viscous fluid that resist bending. The deformation of vesicles and their interaction with the Stokesian fluid play an important role in many biological phenomena. They are used to understand the properties of biomembranes [28, 63] and to simulate the motion of blood cells, in which vesicles with moderate viscosity contrast are used to model red blood cells and high viscosity contrast vesicles or rigid particles are used to model white blood cells [9]. Nonetheless, our contact algorithms do not make any assumptions about the nature of the particles and are applicable to deformable particles with any constitutive properties.

For fluid flow, we use the boundary integral formulation of the Stokes equations. It offers a natural approach for accurate simulation of vesicle flows by reducing the problem to solving equations on surfaces and eliminating the need for discretizing rapidly evolving 3D fluid volumes. However, in non-dilute suspensions, these methods are hindered by a number of difficulties, such as inaccuracies in computing near-singular integrals and artificial force singularities caused by non-physical intersection of particles. In the case of high volume-fraction suspensions, numerical difficulties related to contact and near-contact are of particular importance.

The dynamics of particle collision in Stokes flow is governed by lubrication film formation and drainage, which has a time scale much shorter than that of the flow [27]. In principle, in an accurately resolved flow, fluid forces prevent the contact between particles. However, solely relying on the hydrodynamics to prevent contact requires the accurate solution of the flow in the lubrication film, which in turn entails extremely fine spatial and temporal resolution accompanied by increasingly ill-conditioned linear systems in the boundary integral setting [85, 88] — becoming computationally impractical as the volume fraction increases.

Introducing artificial repulsion forces along with adaptive time stepping [59, 82, 83] is effective at maintaining stability and efficiency in dilute suspensions. However, the time-step in this case is determined by the closest pair of vesicles, and tends to be uniformly small for dense suspensions (see Section 3.4.5). Furthermore, the heuristic parameters of the repulsion force may be difficult to determine automatically, and often require problem-dependent adjustments (Section 3.4.5 and Table 3.5). This is insufficient for a general platform for simulating vesicle dynamics.

Our contributions In this work, we take a different approach: we augment the governing equations with a collision-free constraint, i.e., Eq. (3.2.8). For the model we consider in this chapter, such a constraint is physically redundant, as non-penetration is ensured by fluid forces for a sufficiently high resolution. However, in the numerical context, it plays an important role by ensuring both robustness and accuracy of simulations for a given discretization accuracy.

Contact constraints ensure that the geometry remains intersection-free, even for relatively coarse spatial and temporal discretizations, where the fidelity of the numerical model is insufficient for resolving the lubrication film precisely enough to prevent contact.

One of our principal contributions is the distributed parallel version of contact detection and resolution algorithms. We describe an efficient fully-parallel contact detection algorithm based on fast parallel sorting [98]. Our algorithm utilizes an implicit grid, sorted in Morton curve order on distributed memory machines, for inter-process collision detection, and uses an explicit spatial grid to perform intra-process collision detection calculation locally. The contact constraints are formulated as a Nonlinear Complementarity Problem (NCP). The solution of the NCP problem and the construction of the necessary matrices are both done in parallel based on the iterative minimal map Newton method.

We accelerate the computation of far-field hydrodynamics interactions using the highly optimized PVFMM library [56]. The PVFMM library supports periodic boundary conditions and this allows us to simulate vesicles in periodic flows with a prescribed volume fraction.

Our method permits simulations with high volume fractions (we report results with up to 60% volume fraction, e.g., Fig. 3.8). Within our framework, the time step size is independent from the volume fraction and the simulation wall-clock-time is at least an order of magnitude faster than the adaptive case (Section 3.4.5).

Related work This chapter is most closely related to [53, 59, 88]. We extend these works to enable long-time contact-aware simulation of concentrated vesicle suspensions in parallel. In [53], we proposed the initial version of contact detection and resolution algorithms for 2D and demonstrated that these algorithms enable long-term simulations, increase robustness, and reduce the computational costs. In [59], several novel computational algorithms (e.g., adaptive time stepping, robust near-singular integration, and adaptive mesh refinement) are proposed to facilitate long-time simulation of concentrated suspensions. A short-range repulsion force along with adaptive time stepping were introduced to avoid collision, permitting simulations with up to 35% volume fraction. If a collision is detected, the time step is backtracked and time step size is refined, which in high volume-fraction cases, may result in very small time steps (Section 3.4.5). Our method enables higher density simulations with significantly larger time steps, and does not require parameter tuning for the penalty function.

More broadly, Stokesian particle models are employed to theoretically and computationally investigate the properties of biological membranes [92], drug-carrying capsules [97], and blood cells [67, 76]. There is an extensive body of work on numerical methods for Stokesian particulate flows and a review of the literature up to 2001 can be found in [78]. Reviews of later advances can be found in [86, 88, 103]. Here, we briefly summarize some notable numerical methods and discuss the most recent developments.

Integral equation methods have been used extensively for the simulation of Stokesian particulate flows such as droplets and bubbles [51, 52, 89, 123], vesicles [22, 25, 76, 86, 88, 94, 103, 120, 121], and rigid particles [74, 75, 119]. Other methods — such as phase-field approach [11, 18], immersed boundary and front tracking methods [43, 113], and level set method [47] — are used by several authors for the simulation of particulate flows.

In Chapter 2 we extensively reviewed works on collision detection and handling in the context of (i) Stokesian flows [25, 51, 68, 83, 93, 122, 124, 125]; (ii) contact mechanics and response [24, 39, 46, 81, 102, 109, 110]; and (iii) computer graphics [8, 19, 23, 34, 35, 36, 69, 79] . We refer the interested reader to [53] for more details; here, we focus on most closely related work.

Our constraint-handling algorithms belongs to a large family of *constraint-based methods*, commonly used to handle contacts reliably in many applications, primarily in computer graphics. This class of methods meets our goals of providing robustness and improving efficiency of contact response, while minimizing the impact on the physics of the system. Our contact resolution approach is directly based on [36] and is closest to [2], in which the intersection volume and its gradient with respect to control vertices are computed at the candidate step. The non-collision is enforced as a constraint on this volume, which leads to a much smaller system compared to distance formulation between geometric primitives. The constrained formulation leads to an LCP problem. [36] assumes linear trajectory between edits and defines a *space-time interference volume* (STIV) which serves as a gap function. We use a similar formulation to define our contact constraint.

Parallel algorithms for collision detection, for shared and distributed memory architectures, often include two stages: the first stage is culling, i.e., reducing the set of potential collision pairs using a fast and conservative criterion to determine which pairs do no intersect, followed by precise collision detection between remaining pairs. Many authors used multi-threading and GPU computation to accelerate different stages of collision detection [50, 61, 99, 100, 101]. The main focus of these algorithms is distributing the task of collision culling between threads and they use a variety of techniques.

[50] proposed a GPU-based discrete collision detection algorithm, in which axis-aligned bounding boxes are computed for each object followed by sweep-and-prune steps on the GPU to identify a small set of collision candidates efficiently. In [61], surfaces are approximated by large collections of padded spheres and intersections between spheres are used to cull collision candidates. [41] developed a parallel continuous collision detection algorithm for heterogeneous shared-memory architectures. The algorithm uses bounding volume hierarchy (BVH) for culling. CPUs perform the BVH traversal and culling, while GPUs perform collision tests between geometric primitives.

For distributed memory collision handling, more complex data structures are required. [38] presents a method for rigid body simulation on distributed memory machines. The domain of interest is partitioned and distributed over all MPI processes and each process maintains a list of objects in its domain.

The algorithms we present in this work shares a number of features with [17, 70, 108]. [108] presents a parallel distributed-memory N-Body algorithm. This algorithm uses a parallel octree using the Morton ordering curve and a local essential tree to form a distributed memory octree for N-body interaction calculation. [17] describes a parallel continuous collision detection algorithm for rigid bodies.

Rigid bodies are approximated by spherical bounding volumes and space-time axis aligned bounding boxes are computed for each sphere. The domain is divided into cells, each with a list of overlapping rigid bodies. The cells are distributed over MPI processes and the lists are dynamically updated in parallel. Within each cell, the collision detection is performed locally. Similarly, [70] (a shared-memory algorithm) use spatial cells along with hashing for candidate collision identification, followed by primitive collision tests.

3.1.1 Nomenclature

In Table 3.1, we list symbols and operators used in this chapter. Lowercase letters refer to scalars, and lowercase bold letters refer to vectors. Discretized quantities are denoted by sans serif letters.

3.1.2 Synopsis of the method

We use the boundary integral formulation based on [88]. The basic formulation uses integral equation form of the problem and includes the effects of the viscosity contrast. We add contact constraints to this formulation as an inequality constraint on a gap function that is based on *space-time intersection volume* (Section 3.2). The contact force is then parallel to the gradient of this volume with the Lagrange multiplier as its magnitude.

We solve the resulting contact NCP for the Lagrange multipliers of the constraints using a Newton-like matrix-free method as a sequence of Linear Complementarity Problems (LCP) [15, 21]. Each LCP is solved iteratively using GMRES. The spherical harmonics bases are used for spatial discretization. For time stepping, we use semi-implicit backward Euler (Section 3.3).

Symbol	Definition
γ_i	The boundary of the i^{th} vesicle
γ	$\cup_i \gamma_i$
μ	Viscosity of the ambient fluid
μ_i	Viscosity of the fluid inside i^{th} vesicle
ν_i	The viscosity contrast μ_i/μ
σ	Tension
χ	Shear rate
ϖ_i	The domain enclosed by γ_i
ϖ	$\cup_i \varpi_i$
$\mathcal{G}$	Stokes Single-layer operator
$\mathcal{T}$	Stokes Double-layer operator
LCP	Linear Complementarity problem
NCP	Nonlinear Complementarity Problem
STIV	Space-Time Interference Volumes

Symbol	Definition
LI	Locally Implicit time stepping
CLI	*Constrained* Locally Implicit time stepping
GI	Globally Implicit time stepping
RGI	Repulsion-based Globally Implicit time stepping
d_m	Minimum separation distance
$\boldsymbol{f}_\sigma$	Tensile force
$\boldsymbol{f}_b$	Bending force
$\boldsymbol{f}_c$	Collision force
J	Jacobian of contact volumes V
$\boldsymbol{n}$	Unit outward normal
$\boldsymbol{u}$	Velocity
$\boldsymbol{u}^\infty$	The background velocity field
V	Contact volumes
$\boldsymbol{X}$	A Lagrangian point on a surface

Table 3.1: Index of frequently used symbols, operators, and abbreviations.

In Section 3.3, we present a summary of the spatial and temporal discretization. We also present parallel algorithms for collision culling, STIV computation, and the solution of LCP's.

In Section 3.4, we present results showing the accuracy and effectiveness of our scheme. We also present weak and strong scaling results for simulations on up to 16K processors and report wall-clock-time for different methods and volume fractions.

3.2 Formulation

In this section, we briefly outline mathematical formulation for suspension of vesicles in Stokes flow with contact constraints. We presented a more in-depth formulation applicable to 2D and 3D vesicle flows in [53, 88].

We consider the Stokes flow with N vesicles suspended in a Newtonian fluid. The fluid domain is assumed to be unbounded. In Stokesian flow, due to high viscosity and/or small length scale, the ratio of inertial and viscous forces (the Reynolds number) is small and the fluid flow can be described by the incompressible Stokes equation:

$$-\mu\Delta \boldsymbol{u}(\boldsymbol{x}) + \nabla p(\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x}) \quad \text{and} \quad \nabla\cdot \boldsymbol{u}(\boldsymbol{x}) = 0 \qquad (\boldsymbol{x} \in \mathbb{R}^3), \tag{3.2.1}$$

$$\boldsymbol{F}(\boldsymbol{x}) = \int_\gamma \boldsymbol{f}(\boldsymbol{X})\delta(\boldsymbol{x}-\boldsymbol{X})\,\mathrm{d}A(\boldsymbol{X}), \tag{3.2.2}$$

where $\boldsymbol{f}$ is the surface density of the force exerted by the vesicles' membrane on the fluid and μ denotes the viscosity of ambient fluid. We use $\boldsymbol{x}$ to denote an Eulerian point in the fluid ($\boldsymbol{x} \in \mathbb{R}^3$) and $\boldsymbol{X}$ to denote a Lagrangian point on the vesicles. We let γ_i denote the boundary of the i^{th} vesicle ($i = 1, \ldots, N$), ϖ_i denote the domain enclosed by γ_i, μ_i denote viscosity of the fluid inside that vesicle, and $\gamma := \bigcup_i \gamma_i$. Equation (3.2.1) is valid for $\boldsymbol{x} \in \varpi_i$ by replacing μ with μ_i. The governing equations are augmented with the no-slip boundary condition on the surface of vesicles

$$\boldsymbol{u}(\boldsymbol{X}, t) = \boldsymbol{X}_t \qquad (\boldsymbol{X} \in \gamma), \tag{3.2.3}$$

where $\boldsymbol{X}_t := \frac{\partial \boldsymbol{X}}{\partial t}$ is the material velocity of a point $\boldsymbol{X}$ on the surface of the vesicles.

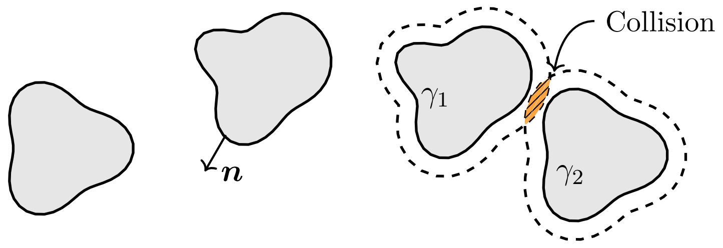


Figure 3.1: Schematic. *Vesicles are suspended in free-space, both filled with fluid. The vesicle boundaries are denoted by γ_i $(i = 1, \dots, N)$. The outward normal vector to the boundaries is denotes by $\boldsymbol{n}$. The dotted lines around boundaries denote the prescribed minimum separation distance for each of them. The minimum separation distance is a parameter and can be set to zero. In this schematic, vesicles γ_1 and γ_2 are in contact. A slice of the space-time intersection volume at the current instance is marked by orange area.*

We assume that the vesicle membrane is inextensible, i.e.,

$$\nabla_\gamma \cdot \boldsymbol{u}(\boldsymbol{X}) = 0 \qquad (\boldsymbol{X} \in \gamma), \tag{3.2.4}$$

where $\nabla_\gamma \cdot$ denotes the surface divergence operator.

3.2.1 Contact definition

It is known [27, 65] that the exact solution of the equations of motion (Eqs. (3.2.1), (3.2.3), and (3.2.4)) keeps particles apart in finite time due to formation of the lubrication film. As a consequence, it is theoretically sufficient to solve the equations with an adequate degree of accuracy to avoid any problems related to overlaps between particles. Nonetheless, achieving this accuracy for many types of flows (most notably, flows with high volume fraction of particles or with complex boundaries) may be prohibitively expensive.

With inadequate computational accuracy, particles may collide with each other or with boundaries (if present). Depending on the numerical method employed, the consequences of this may vary. For methods based on integral equations, the consequences are particularly dramatic, as overlapping boundaries may lead to divergent integrals. To address this issue, we augment the governing equations with a contact constraint, formally written as

$$V(\gamma, t) \geq 0, \tag{3.2.5}$$

where γ denotes the boundary of all particles. The function V is chosen in such a way that $V < 0$ implies some parts of the surface γ are at a distance less than a user-specified constant d_m. Function V may be a vector-valued function, for which the inequality is understood component-wise. This constraint ensures that the suspension remains contact-free independent of numerical resolution.

For the constraint function V, in addition to the basic condition above, we choose a function that satisfies several additional criteria:

(i) V introduces a relatively small number of additional constraints, and

(ii) when V is discretized, no contacts are missed even for large time steps.

To clarify the second condition, suppose we have a small particle rapidly moving towards a planar boundary. For a large time step, it may move to the other side of the boundary in a single step, so any condition that considers an instantaneous quantity depending on only the current position is likely to miss the contact.

To this end, we extend the *Space-Time Interference Volumes* (STIV) from [36] to define the function V^C as the space-time volume swept by the intersecting segments of the boundary over time. To be more precise, for each point $\boldsymbol{X}(\phi, \theta, t_0)$ on the boundary, consider a trajectory $\boldsymbol{X}(\phi, \theta, \tau)$, between a time t_0, for which there are no collisions, and a time t. Points $\boldsymbol{X}(\phi, \theta, \tau)$ define a deformed boundary

$\gamma(\tau)$ for each τ. For each point $\boldsymbol{X}(\phi,\theta,\tau)$, we define $\tau_I(\phi,\theta)$, $t_0 \le \tau_I \le t$, to be the first instance for which this point comes into contact with a different point of $\gamma(\tau_I)$. Assuming an interference-free configuration at t_0, the space-time volume constraint for the time interval $[t_0, t]$ is

$$V^C(\gamma, t) = -\int_{\gamma(t_0)} \int_{\tau_I(\phi,\theta)}^{t} \sqrt{\epsilon^2 + (\boldsymbol{X}_t(\phi,\theta,\tau)\cdot \boldsymbol{n}(\phi,\theta,\tau))^2}\,\mathrm{d}\tau\,\mathrm{d}A, \tag{3.2.6}$$

where $\boldsymbol{n}(\phi,\theta,\tau)$ denotes the normal to $\gamma(\tau)$ at $\boldsymbol{X}(\phi,\theta,\tau)$. The integration is over the initial boundary $\gamma(t_0)$, and we use the fact that the surface is inextensible and the surface metric does not change. Comparing Eq. (3.2.6) with its 2D counterpart [53, Eq. 8], we see that the only difference is that Eq. (3.2.6) is a surface integral. The derivation for Eq. (3.2.6) is given in Section 3.2.2. Writing the constraint for infinitesimal time to obtain the volume constraint V, using the no-slip condition on surface, and taking the variation with respect to the velocity we have

$$\mathrm{d}_{\boldsymbol{u}} V[\delta\boldsymbol{u}] = -\int_{\gamma(t)} \frac{(\boldsymbol{n}\cdot\boldsymbol{u})(\boldsymbol{n}\cdot\delta\boldsymbol{u})}{\sqrt{\epsilon^2 + (\boldsymbol{u}\cdot\boldsymbol{n})^2}} \mathbb{I}_{\text{contact}}\,\mathrm{d}A, \tag{3.2.7}$$

where $\mathbb{I}_{\text{contact}}$ is the indicator function for points that are in contact. We consider each connected component of this volume in Eq. (3.2.6) as a separate volume and impose an inequality constraint on each; while keeping a single volume is equivalent in principle, using multiple volumes avoids undesirable effects in contact resolution computation [36]. We define $V(\gamma, t)$ to be a time-dependent, variable-size vector of contact volumes, with one component per connected component of the contact zone. Depending on the context, we may omit the dependence of V on γ and write $V(t)$ as the contact volume function, or $V(\gamma_i, t)$ for the sub-vector of $V(\gamma, t)$ involving surface γ_i.

In practice, it is desirable to control the minimal distance between particles. Therefore, we define a minimum separation distance $d_m \geq 0$ and modify the constraint such that particles are in contact when they are within d_m distance from each other, as shown in Fig. 3.1. The contact volume with minimum separation distance is calculated with the surface displaced by d_m, i.e., the time t_I is obtained not from the first contact with $\gamma(\tau)$ but rather the first time when surfaces are at distance d_m. Maintaining minimum separation distance — rather than considering pure contact only — eliminates a potentially expensive computation of nearly singular integrals close to the surface and improves the accuracy in semi-explicit time-stepping.

3.2.2 Space-time volume

The space-time volume is a 3D volume embedded in 4D space parameterized by ϕ, θ and t, that is, by the map

$$\boldsymbol{P}(\phi,\theta,t) = (x(\phi,\theta,t), y(\phi,\theta,t), z(\phi,\theta,t), \epsilon t) = (\boldsymbol{X}, \epsilon t),$$

where ϵ is a scaling factor and has the unit of velocity. It is used to make the units consistent. Let $g_{\alpha\beta}$ and $h_{\alpha\beta}$ respectively denote the 3D and 4D metric tensors, i.e.,

$$g_{\alpha\beta} = \boldsymbol{X}_\alpha \cdot \boldsymbol{X}_\beta, \quad \alpha,\beta \in \{\phi,\theta\},$$
$$h_{\alpha\beta} = \boldsymbol{P}_\alpha \cdot \boldsymbol{P}_\beta, \quad \alpha,\beta \in \{\phi,\theta,t\},$$

and $g = \det(g_{\alpha\beta}), h = \det(h_{\alpha\beta})$. Elements $h_{\alpha\beta}$ are related to $g_{\alpha\beta}$, spatial velocity, and basis of tangent space as

$$h = \det(h_{\alpha\beta}) = \det \begin{bmatrix} g_{\phi\phi} & g_{\phi\theta} & \boldsymbol{u} \cdot \boldsymbol{X}_\phi \\ g_{\theta\phi} & g_{\theta\theta} & \boldsymbol{u} \cdot \boldsymbol{X}_\theta \\ \boldsymbol{u} \cdot \boldsymbol{X}_\phi & \boldsymbol{u} \cdot \boldsymbol{X}_\theta & \epsilon^2 + \boldsymbol{u} \cdot \boldsymbol{u}. \end{bmatrix}.$$

Expanding and simplifying the expression for the determinant, we have

$$h = g\Big[\epsilon^2 + (\boldsymbol{u} \cdot \boldsymbol{u}) - (\boldsymbol{u} \cdot \boldsymbol{X}^\alpha)(\boldsymbol{u} \cdot \boldsymbol{X}_\alpha)\Big],$$

where $\boldsymbol{X}^\alpha := g^{\alpha\beta}\boldsymbol{X}_\beta$ is the contravariant basis and $g^{\alpha\beta}$ denotes the dual tensor to $g_{\alpha\beta}$. By the definition of the contravariant basis, we can decompose the velocity as

$$\boldsymbol{u} = (\boldsymbol{u} \cdot \boldsymbol{X}^\alpha)\boldsymbol{X}_\alpha + (\boldsymbol{u} \cdot \boldsymbol{n})\boldsymbol{n}.$$

This in turn implies,

$$\boldsymbol{u} \cdot \boldsymbol{u} = (\boldsymbol{u} \cdot \boldsymbol{X}^\alpha)(\boldsymbol{u} \cdot \boldsymbol{X}^\beta)\boldsymbol{X}_\alpha \cdot \boldsymbol{X}_\beta + (\boldsymbol{u} \cdot \boldsymbol{n})^2 = (\boldsymbol{u} \cdot \boldsymbol{X}^\alpha)(\boldsymbol{u} \cdot \boldsymbol{X}_\alpha) + (\boldsymbol{u} \cdot \boldsymbol{n})^2.$$

Combining these results, we have

$$h = g\Big[\epsilon^2 + (\boldsymbol{u} \cdot \boldsymbol{n})^2\Big].$$

The space-time volume is now easily computed by

$$V^C = \int \sqrt{h}\,\mathrm{d}\phi\,\mathrm{d}\theta\,\mathrm{d}t = \int \sqrt{g(\epsilon^2 + (\boldsymbol{u}\cdot\boldsymbol{n})^2)}\,\mathrm{d}\phi\,\mathrm{d}\theta\,\mathrm{d}t = \int \sqrt{\epsilon^2 + (\boldsymbol{u}\cdot\boldsymbol{n})^2}\,\mathrm{d}A\,\mathrm{d}t,$$

where we substituted $\mathrm{d}A = \sqrt{g}\,\mathrm{d}\theta\,\mathrm{d}\phi$.

3.2.3 Contact constraint

We use the Lagrange multiplier method (e.g., [110]) to enforce contact constraints in the system. While it is computationally more expensive than adding a penalty force for the constraint (effectively, an artificial repulsion force), it has the advantages of (i) eliminating the need to tune the parameters of the penalty force to ensure that the constraint is satisfied; and (ii) keeping nonphysical contact forces introduced into the system to the minimum The constrained system can be written as

$$\begin{aligned} &\min \int_{\mathbb{R}^3} \left(\frac{1}{2}\mu\,\nabla\boldsymbol{u}\cdot\nabla\boldsymbol{u} - \boldsymbol{u}\cdot\boldsymbol{F}\right)\mathrm{d}V, \qquad (3.2.8)\\ &\text{subject to:}\qquad \nabla\cdot\boldsymbol{u}(\boldsymbol{x}) = 0 \quad (\boldsymbol{x}\in\mathbb{R}^3),\\ &\qquad\qquad\qquad \nabla_\gamma\cdot\boldsymbol{u} = 0 \quad (\boldsymbol{X}\in\gamma),\\ &\qquad\qquad\qquad V(\gamma,t) \geq 0. \end{aligned}$$

If we omit the inequality constraint, the remaining three equations are equivalent to the Stokes equations (3.2.1). The Lagrangian for this system is

$$\mathcal{L}(\boldsymbol{u}, p, \sigma, \lambda) = \int_{\mathbb{R}^3} \left(\frac{1}{2}\mu \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \boldsymbol{u} \cdot \boldsymbol{F} - p \nabla \cdot \boldsymbol{u} \right) \mathrm{d}V + \int_\gamma \sigma \nabla_\gamma \cdot \boldsymbol{u} \, \mathrm{d}A + \lambda \cdot V. \tag{3.2.9}$$

The first-order optimality (KKT) conditions yield the following modified Stokes equation, along with the constraints listed in Eq. (3.2.8):

$$-\mu \Delta \boldsymbol{u} + \nabla p = \boldsymbol{F}', \tag{3.2.10}$$

$$\boldsymbol{F}'(\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x}) + \int_\gamma (\boldsymbol{f}_\sigma + \boldsymbol{f}_c)(\boldsymbol{X}) \delta(\boldsymbol{x} - \boldsymbol{X}) \, \mathrm{d}A, \tag{3.2.11}$$

$$\boldsymbol{f}_\sigma = \sigma \Delta_\gamma \boldsymbol{X} + \nabla_\gamma \sigma, \tag{3.2.12}$$

$$\boldsymbol{f}_c = (\, \mathrm{d}_{\boldsymbol{u}} V)^T \lambda, \tag{3.2.13}$$

$$\lambda \geq 0; \lambda \cdot V = 0, \tag{3.2.14}$$

where $\mathrm{d}_{\boldsymbol{u}} V$ is the variation of of V with respect to $\boldsymbol{u}$. The last condition is the complementarity condition — either an equality constraint is active ($V_i = 0$) or its Lagrange multiplier is zero. As we will see in the next section and based on Eq. (3.2.11), the collision force $\boldsymbol{f}_c$ is added to the traction jump across the vesicle's interface.

It is customary to combine $V \geq 0, \lambda \geq 0$, and $\lambda \cdot V = 0$, into one expression and write

$$0 \leq V(t) \quad \perp \quad \lambda \geq 0. \tag{3.2.15}$$

3.2.4 Boundary integral formulation

Following the standard approach of potential theory [74], one can express the solution of the Stokes boundary value problem, Eq. (3.2.10), as a system of singular integro-differential equations on all immersed and bounding surfaces.

The Stokeslet tensor $\mathbf{G}$ and the Stresslet tensor $\mathbf{T}$ are the fundamental solutions of the Stokes equation given by

$$\mathbf{G}(\boldsymbol{r}) = \frac{1}{8\pi\mu}\frac{1}{\|\boldsymbol{r}\|}\left(\boldsymbol{I} + \frac{\boldsymbol{r}\otimes\boldsymbol{r}}{\|\boldsymbol{r}\|^2}\right), \qquad \mathbf{T}(\boldsymbol{r}) = -\frac{3}{4\pi\mu}\frac{\boldsymbol{r}\otimes\boldsymbol{r}\otimes\boldsymbol{r}}{\|\boldsymbol{r}\|^5}.$$

The solution of Eq. (3.2.10) can be expressed by the combination of single- and double-layer integrals. We denote the single-layer integral on the vesicle surface γ_i by

$$\mathcal{G}_{\gamma_i}[\boldsymbol{f}](\boldsymbol{x}) := \int_{\gamma_i} \mathbf{G}(\boldsymbol{x}-\boldsymbol{Y})\cdot\boldsymbol{f}(\boldsymbol{Y})\,\mathrm{d}A, \tag{3.2.16}$$

where $\boldsymbol{f}$ is an appropriately defined density. The double-layer integral is defined as

$$\mathcal{T}_{\gamma_i}[\boldsymbol{q}](\boldsymbol{x}) := \int_{\gamma_i} \boldsymbol{n}(\boldsymbol{Y})\cdot\mathbf{T}(\boldsymbol{x}-\boldsymbol{Y})\cdot\boldsymbol{q}(\boldsymbol{Y})\,\mathrm{d}A, \tag{3.2.17}$$

where $\boldsymbol{n}$ denotes the outward normal (as shown in Fig. 3.1) to the surface γ_i and $\boldsymbol{q}$ is the appropriately defined density. When the evaluation point $\boldsymbol{x}$ is on the integration surface, Eq. (3.2.16) is a singular integral, and Eq. (3.2.17) is interpreted in the principal value sense.

The velocity at a point $\boldsymbol{x}\in\mathbb{R}^3$ can be expressed as the superposition of velocities due to each vesicle

$$\alpha \boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}^{\infty}(\boldsymbol{x}) + \sum_i \boldsymbol{u}^i(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^3, \quad \alpha = \begin{cases} 1 & \boldsymbol{x} \in \mathbb{R}^3 \backslash \gamma, \\ \nu_i & \boldsymbol{x} \in \varpi_i, \\ (1+\nu_i)/2 & \boldsymbol{x} \in \gamma_i, \end{cases} \tag{3.2.18}$$

where $\boldsymbol{u}^{\infty}(\boldsymbol{x})$ represent the background velocity field (for unbounded flows), $\nu_i = \mu_i/\mu$ denotes the viscosity contrast of the i^{th} vesicle, and $\boldsymbol{u}^i$ denotes the velocity contributions from vesicle i. To simplify the representation, we introduce the complementary velocity for each vesicle, defined as $\bar{\boldsymbol{u}}^i = \alpha \boldsymbol{u} - \boldsymbol{u}^i$.

The velocity induced by the i^{th} vesicle is expressed as an integral [77]:

$$\boldsymbol{u}^i(\boldsymbol{x}) = \mathcal{G}_{\gamma_i}[\boldsymbol{f}](\boldsymbol{x}) + (1-\nu_i)\mathcal{T}_{\gamma_i}[\boldsymbol{u}](\boldsymbol{x}) \qquad (\boldsymbol{x} \in \mathbb{R}^3), \tag{3.2.19}$$

where the double-layer density $\boldsymbol{u}$ is the total velocity from Eq. (3.2.18) and $\boldsymbol{f}$ is the traction jump across the vesicle membrane. Based on Eq. (3.2.11), the traction jump is equal to the sum of bending, tensile, and collision forces

$$\boldsymbol{f}(\boldsymbol{X}) = \boldsymbol{f}_b + \boldsymbol{f}_\sigma + \boldsymbol{f}_c \qquad (\boldsymbol{X} \in \gamma). \tag{3.2.20}$$

Note that Eq. (3.2.19) is the contribution from each vesicle to the velocity field. To obtain an equation for the interfacial velocity, Eq. (3.2.19) is substituted into Eq. (3.2.18):

$$\frac{(1+\nu_i)}{2}\boldsymbol{u}(\boldsymbol{X}) = \bar{\boldsymbol{u}}^i(\boldsymbol{X}) + \mathcal{G}_{\gamma_i}[\boldsymbol{f}](\boldsymbol{X}) + (1-\nu_i)\mathcal{T}_{\gamma_i}[\boldsymbol{u}](\boldsymbol{X}) \qquad (\boldsymbol{X} \in \gamma_i), \tag{3.2.21}$$

subject to local inextensibility

$$\nabla_\gamma \cdot \boldsymbol{u} = 0 \qquad (\boldsymbol{X} \in \gamma). \tag{3.2.22}$$

3.2.5 Formulation summary

The formulae outlined above govern the evolution of the suspension. The flow constituents are hydrodynamically coupled through the complementary velocity (i.e., the velocity from all other constituents). Given the configuration of the suspension, the unknowns are velocity $\boldsymbol{u}(\boldsymbol{X})$ and tension σ of vesicles' interface determined by Eqs. (3.2.20–3.2.22). The velocity is integrated for the vesicles' trajectory using Eq. (3.2.3).

This system is constrained by Signorini (KKT) conditions for the contact, Eq. (3.2.15), which is used to compute λ, the strength of the contact force. In the referenced equations above, the complementary velocity is combination of velocities given in Eq. (3.2.19).

3.3 Numerical algorithms

In this section, we describe the algorithms required for solving the dynamics of particulate Stokesian suspensions. Our spatial/temporal representation and surface quadratures follow our previous work [58, 88, 105].

We present a set of parallel distributed-memory algorithms for contact resolution in 3D. The parallelization is a necessity for simulations of reasonable size in 3D. The overall approach follows our two-dimensional contact-aware scheme presented in [53], but new algorithms are needed for the distributed-memory contact detection and resolution.

At every time step, we resolve contacts by solving a nonlinear complementarity problem (NCP) in parallel. The NCP is solved iteratively by recursive linearization and using a new parallel Linear Complementarity Problem (LCP) solver.

In the following sections, we first summarize a brief description of the spatial discretization from [88], then discuss the time discretization with contact constraint and the corresponding parallel algorithms.

3.3.1 Spatial discretization

3.3.1.1 Spherical harmonic expansion

We assume that the boundary of each vesicle is parameterized by a smooth map $\boldsymbol{X}(\phi, \theta)$ from a unit sphere $\mathbb{S}^2$ to $\mathbb{R}^3$ and use the spherical harmonics expansions to represent the surface [105]. The surface of the unit sphere $\mathbb{S}^2$ is parameterized by the spherical angles $(\phi, \theta) \in [0, \pi] \times [0, 2\pi)$. To approximate a function f on γ, we use the spherical harmonic basis Y_{nm} up to degree q

$$f(\phi, \theta) \approx \sum_{n=0}^{q} \sum_{m=-n}^{n} \widehat{f}_{nm} Y_{nm}(\phi, \theta), \quad \text{with} \quad \widehat{f}_{nm} = (f, Y_{nm}) = \int_{\mathbb{S}^2} f \overline{Y}_{nm} \, \mathrm{d}s. \quad (3.3.1)$$

where $\widehat{f}_{nm}$ are the coefficients in the spherical harmonic expansion. A *q-grid* is the grid associated with a q^{th} order spherical harmonic expansion, with $q+1$ Legendre points in the latitude direction and $2q$ equi-spaced points in the longitude direction.

All functions on the surface of vesicle (such as surface position/velocity and tensile/bending/contact forces) are presented in the spherical harmonic basis. Interpolation to collocation points of a finer/coarser grid is done by padding/truncation of the spherical harmonics expansion. To evaluate a function on a non-grid point, we construct local third order interpolants on a 4×4 set of points from the q-grid and interpolate at the target [88].

3.3.1.2 Stokes layer potentials singular and near-singular integration

We need to evaluate single- and double-layer potential in our boundary integral formulation, we use the same quadrature rule for single- and double-layer potential. Consider the single-layer potential $\mathcal{G}_\gamma[\boldsymbol{f}](\boldsymbol{Y})$ from a single surface γ at a target point $\boldsymbol{Y}$. We can express the integral as

$$\mathcal{G}_\gamma[\boldsymbol{f}](\boldsymbol{Y}) = \int_{\phi=0}^{\pi} \int_{\theta=0}^{2\pi} \mathbf{G}(\boldsymbol{Y}, \boldsymbol{X}(\phi,\theta))\ \boldsymbol{f}(\phi,\theta)\ W(\phi,\theta)\ \mathrm{d}\phi\ \mathrm{d}\theta, \tag{3.3.2}$$

where $W(\phi,\theta) = \sqrt{EG - F^2}$ is the area element of the surface (with E, F and G denoting the coefficients of the first fundamental form of γ).

For the target point $\boldsymbol{Y}$ not on the surface γ, the integrand is smooth; we use Gauss-Legendre quadrature for ϕ and trapezoidal quadrature for θ directions. We discretize the integral in Eq. (3.3.2) by a quadrature rule over the q-grid

$$\mathcal{G}_\gamma[\boldsymbol{f}](\boldsymbol{Y}) \approx \sum_{i=0}^{q} \sum_{j=0}^{2q-1} \mathsf{G}(\mathsf{Y}, \mathsf{X}_{ij})\ \mathsf{f}_{ij}\ W_{ij}\ w_{ij} \tag{3.3.3}$$

where w_{ij} are the quadrature weights, X_{ij}, f_{ij} and W_{ij} are the surface position, the density function and the area element evaluated on the q-grid, respectively. We use the Fast Multipole Method (FMM) to accelerate the calculation in Eq. (3.3.3). As Y approaches γ, Eq. (3.3.3) is insufficient to accurately evaluate near-singular layer-potential. We use the near-singular integration scheme discussed in [58].

For the target point $\boldsymbol{Y}$ on the surface γ, the layer potential is singular. We use the singular integration algorithm discussed in [58, 105] for both the Stokes single-layer potential and the Stokes double-layer potential. This quadrature scheme involves rotation of the spherical harmonics and is spectrally accurate for both single- and double-layer potentials.

3.3.1.3 Piecewise-linear triangular discretization for constraints

Similar to the 2D case, while the spectral spatial discretization is used for most computations, it poses a problem for the minimal-separation constraint discretization. Computing parametric surface intersections, an essential step in the STIV computation, is relatively expensive and difficult to implement robustly, as this requires solving nonlinear equations for intersections. We follow the same approach as in 2D [53] and use a piecewise-linear triangular discretization of the surface to calculate the space-time contact volume and its gradient.

The collocation points for the q-grid naturally give a quadrilateral mesh of the surface, except at the poles, where quads are degenerate. Splitting each quadrilateral along the diagonal we get a piecewise-linear triangular discretization of the surface as illustrated in Fig. 3.2.

We first upsample the q-grid to a q^{up}-grid and convert the q^{up}-grid to a triangular mesh for STIV calculation. Since we opt for a low-order, piecewise-linear approximation of the vesicle, we need to use an algorithm that ensures that at least the target minimal separation is maintained between actual smooth surfaces represented by spherical harmonics. For the triangular mesh, we set the separation distance to $(1 + 2\alpha)d_m$, where d_m is the target minimum separation distance. We observe that the sensitivity to the separation distance on the overall accuracy is low in most situations as explored in Section 3.4; in practice we choose $\alpha = 0.05$ and we find that $q^{\mathrm{up}} = 32$ is sufficient for $\|\mathbf{X}^{q^{\mathrm{up}}} - \boldsymbol{X}(\phi, \theta)\|_\infty < \alpha d_m$.

To compute the contact constraint with the triangular mesh of a q^{up}-grid, we calculate the discretized space-time contact volume as the sum of triangle-vertex contact volumes $V = \sum_k V_k(\mathbf{t}, \mathbf{X})$, where k indexes triangle-vertex pairs. Parallel algorithms to find colliding triangle-vertex pairs are discussed Section 3.3.3.

For each triangle $\mathbf{t}(\mathbf{X}_{i-1}, \mathbf{X}_i, \mathbf{X}_{i+1})$ and vertex $\mathbf{X}_k$ pair, we solve a degree six equation to find their earliest contact time τ_I assuming linear trajectory between the initial position at time t_n and candidate position at t_{n+1}:

$$\begin{aligned}&([\mathbf{X}_k(t) - \mathbf{X}_{i-1}(t)] \cdot [(\mathbf{X}_i(t) - \mathbf{X}_{i-1}(t)) \times (\mathbf{X}_{i+1}(t) - \mathbf{X}_{i-1}(t))])^2 \\ &\qquad -d_m^2 \left\| \left[(\mathbf{X}_i(t) - \mathbf{X}_{i-1}(t)) \times (\mathbf{X}_{i+1}(t) - \mathbf{X}_{i-1}(t))\right] \right\|^2 = 0,\end{aligned} \tag{3.3.4}$$

where $\mathbf{X}_k(t) = \mathbf{X}_k(t_n) + t\mathbf{U}_k$, with the linear trajectory assumption the vertex velocity is defined as $\mathbf{U}_k = [\mathbf{X}_k(t_{n+1}) - \mathbf{X}_k(t_n)] / \Delta t$. We calculate the triangle-

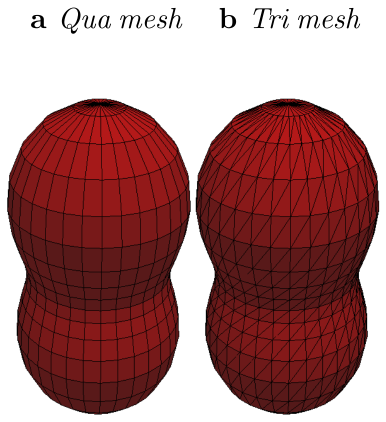


Figure 3.2: *Piecewise-linear triangular discretization. Illustrating the conversion of a* 16*-grid Fig. 3.2a to a piecewise-linear triangular mesh Fig. 3.2b.*

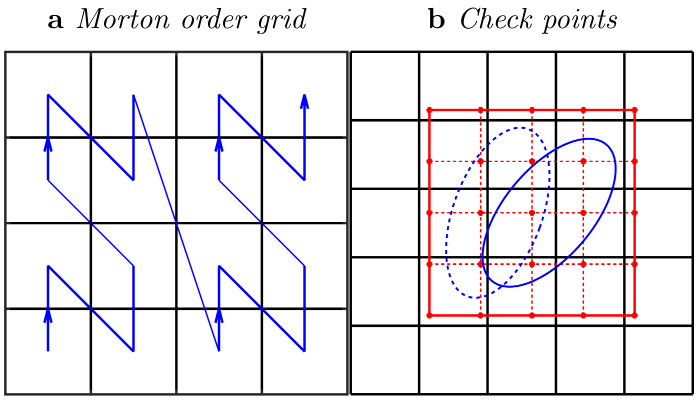


Figure 3.3: Morton order grid. *Diagram Fig. 3.3a shows the* 2D *Morton curve order for a grid. Fig. 3.3b shows the schematic of* 2D *space-time bounding box (red) of a vesicle (dotted blue curve is the initial position, solid blue curve is the candidate position) filled with check points.*

vertex contact volume using Eq. (3.2.6):

$$V_k(\mathbf{t}, \mathbf{X}) = (t - \tau_I)(\epsilon^2 + (\mathbf{U}_k \cdot \mathbf{n}(\tau_I))^2)^{1/2}|\boldsymbol{t}|, \tag{3.3.5}$$

where $\mathbf{n}(\tau_I)$ is the normal to the triangle $\mathbf{t}(\tau_I)$ at the intersection time and $|\mathbf{t}|$ is the area of the triangle. For each triangle-vertex contact volume, we calculate the gradient of Eq. (3.3.5) with respect to the vertices $\mathbf{X}_{i-1}$, $\mathbf{X}_i$, $\mathbf{X}_{i+1}$ and $\mathbf{X}_k$ and by summing over all the triangle-vertex contact pairs, we get the total space-time interference volume and its gradient.

3.3.2 Temporal discretization

Our temporal discretization is based on the locally-implicit time-stepping scheme in [88]: we treat intra-vesicle interactions implicitly and inter-vesicle interactions explicitly. We combine this method with minimal-separation constraint and use first-order backward Euler time stepping (more accurate time-stepping methods such as spectral deferred correction (SDC) method [53] can be easily integrated). We denote this locally-implicit scheme with collision constraint by CLI. In Section 3.4, we compare this method to the same scheme without constraints (LI) and the globally semi-implicit with/without repulsion(GI and RGI) schemes, where all interactions treated implicitly as in [58].

Marking the unknowns to be solved for with '$\cdot^+$' superscript and by denoting the position, velocity and traction jumps at each point of the i^{th} vesicle by $\mathbf{X}_i$, $\mathbf{u}_i^+ = (\mathbf{X}_i^+ - \mathbf{X}_i)/\Delta t$, and $\mathbf{f}_i$, respectively, we obtain the following time-stepping equation:

$$\frac{1+\nu_i}{2}\mathbf{u}_i^+ = \bar{\mathbf{u}}_i + \mathbf{G}_{\gamma_i}\mathbf{f}_i(\mathbf{X}_i^+, \sigma_i^+, \lambda^+) + (1-\nu_i)\mathbf{T}_{\gamma_i}\mathbf{u}_i^+, \tag{3.3.6}$$

$$\nabla_\gamma \cdot \mathbf{u}_i^+ = 0, \tag{3.3.7}$$

$$\mathbf{f}_i(\mathbf{X}_i^+, \sigma_i^+, \lambda^+) = \mathbf{f}_b(\mathbf{X}_i^+) + \mathbf{f}_\sigma(\sigma_i^+, \mathbf{X}_i) + \mathbf{f}_c^+, \tag{3.3.8}$$

$$\mathbf{f}_c^+ = ((\,\mathrm{d}_{\boldsymbol{u}}\mathsf{V}^+)^T\lambda^+)_i, \tag{3.3.9}$$

$$0 \le \mathsf{V}(\gamma; t^+) \quad \perp \quad \lambda^+ \ge 0, \tag{3.3.10}$$

where $\mathsf{V}(\gamma; t^+)$ is the space-time volume.

Let $\mathbf{A}\mathbf{X}^+ = \mathbf{b}$ be the linear system that is solved at each iteration of a CLI scheme. $\mathbf{A}$ is a block diagonal matrix, with blocks $\mathbf{A}_{ii}$ corresponding to the self interactions of the i^{th} vesicle. All inter-vesicle interactions are treated explicitly, and thus included in the right-hand side $\mathbf{b}$. The matrix $\mathbf{A}$ is obtained by combining and rearranging Equations (3.3.6–3.3.9). We write Eq. (3.3.6) in a compact form as

$$\mathbf{A}\mathbf{X}^+ = \mathbf{b} + \mathbf{G}\mathbf{f}_c^+, \tag{3.3.11}$$

$$0 \le \mathsf{V}(\gamma; t^+) \quad \perp \quad \lambda \ge 0. \tag{3.3.12}$$

At each step, we need to resolve contacts by solving this mixed Nonlinear Complementarity Problem (NCP). We outline our parallel algorithms for solving this NCP next.

Algorithm 3: CONTACT-FREE TIME-STEPPING.

```
    input  : $\mathbf{X}, \mathbf{f}_c$
    output: $\mathbf{X}^+, \mathbf{f}_c^+$
 1  $\mathbf{A} \leftarrow \mathbf{A}(\mathbf{X})$
 2  $\mathbf{b} \leftarrow \mathbf{b}(\mathbf{X}, \mathbf{f}_c)$
 3  $\mathbf{f}_c^+ \leftarrow 0$
 4  $k \leftarrow 0$
 5  $\mathbf{X}^\star \leftarrow \mathbf{A}^{-1}\mathbf{b}$                  // Get initial candidate position
 6  $\mathsf{V}, \mathbf{J} \leftarrow$ ContactVolume($\mathbf{X}, \mathbf{X}^\star$)                   // Alg. 5
 7  while $\mathsf{V} < 0$ do
 8      $\mathbf{B} \leftarrow$ FormLCP($\mathbf{J}, \mathbf{A}$)                          // Alg. 6
 9      $\lambda \leftarrow$ LCPSolver($\mathsf{V}, \mathbf{B}$)                        // Alg. 7
10      $k \leftarrow k + 1$
11      $\mathbf{b} \leftarrow \mathbf{b} + \mathbf{G}\mathbf{J}^T\lambda$      // Update the RHS with the new collision
         force
12      $\mathbf{X}^\star \leftarrow \mathbf{A}^{-1}\mathbf{b}$                // Get new candidate position $\mathbf{X}^\star$
13      $\mathsf{V}, \mathbf{J} \leftarrow$ ContactVolume($\mathbf{X}, \mathbf{X}^\star$)      // Check collision for $\mathbf{X}^\star$
14      $\mathbf{f}_c^+ \leftarrow \mathbf{f}_c^+ + \mathbf{J}^T\lambda$             // Accumulate the collision force
15  $\mathbf{X}^+ \leftarrow \mathbf{X}^\star$
```

3.3.2.1 Contact-resolving iterations

To approximately solve Eqs. (3.3.11) and (3.3.12), we iteratively construct linearizations to $\mathsf{V}(\gamma; t)$ around the current candidate position and solve a sequence of LCPs:

$$\mathbf{A}\mathbf{X}^\star = \mathbf{b} + \mathbf{G}\mathbf{J}^T\lambda. \tag{3.3.13}$$

$$0 \leq \mathsf{V}(\gamma; t^{+k}) + \mathbf{J}\Delta\mathbf{X} \quad \perp \quad \lambda \geq 0, \tag{3.3.14}$$

until the original NCP is solved to the desired accuracy. In Eq. (3.3.14), $\mathbf{X}^\star$ is the candidate solution, $\Delta\mathbf{X}$ is the update to get the new candidate solution, and $\mathbf{J}$ denotes the Jacobian of the STIV $\mathrm{d}_X\mathsf{V}(\gamma, t^{+k})$, where k indexes the contact-resolving iterations.

Alg. 3 summarizes the contact-free time-stepping to solve Eqs. (3.3.11) and (3.3.12) as a series of linearization steps in the form of Eqs. (3.3.13) and (3.3.14). In lines 1 to 5, we solve the unconstrained system $\mathbf{AX}^\star = \mathbf{b}$ using the solution from previous time step. In line 6, the STIVs are computed using a parallel collision detection algorithm, which is discussed in Section 3.3.3. The loop in lines 7−14 is the linearized contact-resolving steps. Substituting Eq. (3.3.13) into Eq. (3.3.14), and using the fact that $\Delta\mathbf{X} = \mathbf{A}^{-1}\mathbf{GJ}^T\lambda$ we cast the problem in the standard LCP form

$$0 \leq \mathsf{V} + \mathbf{B}\lambda \quad \perp \quad \lambda \geq 0, \tag{3.3.15}$$

where $\mathbf{B} = \mathbf{JA}^{-1}\mathbf{GJ}^T$. The LCP solver is called on line 9 to obtain the magnitude of the constraint force, which is in turn used to obtain new candidate positions that may or may not satisfy the constraints. In line 11, the collision force is incorporated into the right-hand-side $\mathbf{b}$ for self interaction in the next LCP iteration. Line 13 checks the minimal-separation constraints for the candidate solution. In line 14, the contact force is updated, which will be used to form the right-hand-side $\mathbf{b}$ for the global interaction in the next time step.

3.3.3 Parallel collision handling

In this section, we describe the most challenging algorithmic part of our method, parallel collision handling, which is essential for scalability.

To avoid costly communication and computation, our contact detection is performed in two phases. In the first phase, we find intersecting bounding boxes of particles (Alg. 4). For each particle, this results in a list of other particles that it may be colliding with (i.e., *candidate pairs*). In the second phase, we communicate the mesh information for the particles in the list and compute the pairwise STIVs (Alg. 5). These intersection volumes and their gradients (Alg. 3 lines 6 or 13) are used in the LCP to find the magnitude of contact force.

In our algorithm descriptions, superscript p denotes data that resides on process p. Each vesicle is assigned to a process p and the assignment does not change during the simulation. Variables without superscripts are either shared variables among all processes, or global arrays whose local parts are denoted with the superscripts. We use subscripts for indices of the vesicles, bounding boxes, etc. Table 3.2 summarizes the variables used in this section.

3.3.3.1 Phase 1: Bounding box intersections

To narrow down the set of potential collision pairs efficiently, we initially use *space-time* bounding boxes as collision proxies, i.e., 3D bounding boxes enclosing point trajectories from the initial positions X_0^p to candidate positions X_1^p for each vesicle. Each axis-aligned bounding box is stored as a pair of points $\{\underline{\mathbf{b}}_i^p, \bar{\mathbf{b}}_i^p\}$, that are the corners of the box with lexicographically-ordered minimum and maximum coordinate values.

To identify the intersecting bounding box pairs efficiently and in parallel, we use a spatial grid algorithm. There are three main steps in finding the intersecting bounding box pairs using a spatial grid:

(1) For each bounding box, find grid cells it overlaps.

(2) For each grid cell, compute a list of bounding boxes overlapping it by merging the lists of (box, grid cell) pairs from step 1, and sorting it by grid cell.

(3) For each grid cell with a non-empty list, perform intersection check for all bounding box pairs in that cell and find all the intersecting bounding box

Symbol	Definition
n_p	Number of MPI processes
p	Process index
n	Number of points on a vesicle (assumed fixed)
N^p	Number of vesicles
$\overline{N}^p$	Number of ghost vesicles
N_c^p	Number of check points
N_P^p	Number of candidate vesicle pairs
N_v^p	Number of contact volumes
X_0^p, X_1^p	Initial and candidate positions (size nN^p)
$\overline{X}_0^p, \overline{X}_1^p$	Initial and candidate positions of ghost vesicles
I^p	Set of global indices for vesicles (vectors of size N^p)
$\overline{I}^p$	Set of global indices for ghost vesicles
B^p	Set of space-time bounding boxes of vesicles
I_b^p	Set of global bounding-boxes indices
P^p	Set of index pairs of intersecting bounding boxes, $\{(j_r, k_r) \mid r = 1, \ldots, N_P^p, j_r \in I_b^p, k_r \in I_b, j_r < k_r\}$
$\mathbf{c}^p, I_c^p, I_{c,b}^p, M^p$	Positions, global indices, vesicles indices, and Morton codes of check points
h_m	Morton order grid cell size
V^p, I_v^p	Contact volumes and their global indices (size N_v^p)
J^p	Jacobian of the vector of contact volumes with respect to vesicles (size $N_v^p \times 3nN^p$)
$\overline{J}^p$	Jacobian with respect to ghost vesicles (size $N_v^p \times 3n\overline{N}^p$)
$\lambda^p = \lambda(I_v^p)$	Lagrange multipliers

Table 3.2: Parallel variables. *Superscript p denotes the data corresponding to process p.*

pairs.

Parallel version The most direct approach to parallelizing this algorithm is to distribute the lists of boxes associated with grid cells across processes. However, performing step (2) efficiently is difficult in this case, since merging the lists obtained in step (1) requires irregular communication pattern between all processors.

Instead of using an explicitly distributed spatial grid data structure storing lists

Algorithm 4: Bounding Box Intersection.

```
input  : A set of axis aligned bounding boxes $B^p$, $I_b^p$
output: Intersecting bounding box index pairs: $P^p$
// (1) Insert boxes to the tree by computing the Morton ID
   of grid cells they intersect.
1  $h_m \leftarrow$ average $\left(\|\underline{\mathbf{b}}_i^p - \bar{\mathbf{b}}_i^p\|\right)$          // Parallel reduction
2  $\{\mathbf{c}^p, I_c^p, M^p, I_{c,b}^p, \underline{\mathbf{x}}^p, \bar{\mathbf{x}}^p\} \leftarrow$ generateCheckPts $(B^p, I_b^p, h_m)$
3  Update $I_c^p$ to global index                         // Parallel scan

   // (2) Merge the lists by sorting check points using Morton
      IDs and scatter data based on $I_c^p$.
4  hypercubeQSort $\left(M^p, \{I_c^p, I_{c,b}^p, \underline{\mathbf{x}}^p, \bar{\mathbf{x}}^p\}\right)$

   // (3) Local check of intersecting bounding box pairs
5  $P^p \leftarrow \emptyset$
6  for each $m \in M^p$ do
7      for each bounding box pair $\{j,k\}$ $(j,k \in I_{c,b}^p)$ in grid $m$ do
8          if bbiCheck($\underline{\mathbf{b}}_j, \bar{\mathbf{b}}_j, \underline{\mathbf{b}}_k, \bar{\mathbf{b}}_k$) then
9              Add intersecting bounding box pair$\{j,k\}$ to $P^p$

10 Send $P^p$ $(\{j,k\})$ to the process $q$ such that $j \in I_b^q$
11 $P^p \leftarrow$ unique($P^p$)
```

of boxes, we use an implicit representation of grid cells based on Morton curve order numbering. Morton IDs of cells are assigned to *check points*, sampled on bounding boxes in a way that guarantees that at least one check point is contained in every grid cell overlapping the box. Parallel-sorting the check points by their Morton IDs collects, on each processor, a set of check points corresponding to bounding boxes overlapping the same grid cells, as their Morton IDs will be the same. Then bounding boxes overlapping each grid cell are checked for intersections in parallel. Finally, detected candidate intersection pair lists are scattered back to the processors owning bounding boxes contained in the list. Next, we describe the algorithm more formally. In the pseudocode (Alg. 4) lines 1–3, line 4, and lines

	Process 1					Process 2			
	Unsorted								
M:	111	110	001	000		001	000	000	101
I_c:	1	2	3	4		5	6	7	8
$I_{c,b}$:	1	1	1	1		2	2	2	2
local data:	a	b	c	d		e	f	g	h
	Sorted (shaded rows are misaligned)								
M:	000	000	000	001	001	101	110	111	
I_c:	4	6	7	3	5	8	2	1	
$I_{c,b}$:	1	1	1	1		2	2	2	2
local data:	a	b	c	d		e	f	g	h
	Scattered (data is aligned)								
M:	000	000	000	001	001	101	110	111	
I_c:	4	6	7	3	5	8	2	1	
$I_{c,b}$:	1	2	2	1	2	2	1	1	
local data:	d	f	g	c	e	h	b	a	

Table 3.3: SORTING AND SCATTERING OF CHECK POINTS ON TWO PROCESSES. *A simple check points sorting and data scattering example. This example illustrates the data movement between two processes. Unsorted rows show the input information on process* 1 *and process* 2. *Sorted rows show the sorted check points' Morton codes M and shuffled index set I_c with $I_{c,b}$ and unscattered data on each process. Shaded rows are misaligned. Scattered rows show the result of scattering using the shuffled index set I_c as a scatter mapping from original data to scattering place on each process.*

5–11 respectively are steps (1) to (3) outlined above.

We view the spatial grid as a uniformly refined octree. The depth of octree is $\log(L/h_m)$ where h_m is the grid cell size. We set h_m to the average diagonal length of all vesicle bounding boxes and L is the domain length. Each leaf in the octree is associated with a unique Morton ID, as shown in Fig. 3.3a. Table 3.3 illustrates the data movement in the parallel sorting algorithm(line 4 of Alg. 4).

In step (1), check points $\mathbf{c}^p$ are defined for the bounding boxes B^p (lines 2 and 3). As schematically shown in Fig. 3.3b, check points are located uniformly inside each bounding box with a spacing less than h_m. With this spacing, a bounding box should have at least one check point in an any grid cell it overlaps. Each check point is associated with its originating bounding box index and that bounding box's spatial data, i.e., $I_{c,b}^p$ and $\{\underline{\mathbf{x}}^p, \bar{\mathbf{x}}^p\}$. Our choice of spacing results in 2^3 to 3^3 check points for each bounding box.

We compute the set of all Morton IDs M^p for all check points. Since each Morton ID is uniquely associated with a grid cell and each bounding box has at least one check point inside the grid cell overlapping with that bounding box, our lists M^p include the Morton ID of all grid cells that intersect with each bounding box, possibly multiple times (we drop the duplicates).

Initially, the lists I_c^p are the global indices of the check points on process p; to calculate this global index we perform an MPI scan operation on the number of check points on p^{th} process N_c^p. With the Morton IDs and global check point index computed for each check point, we can move to the next stage.

Step (2) of the algorithm (merging lists) is equivalent to a parallel sort on the Morton IDs followed by scattering of data associated with a check point using the shuffle index I_c^p obtained as a result of the sort.

We use a parallel hypercube quicksort algorithm [98]. The hypercube quicksort algorithm takes the Morton code M^p (keys) and index set of check points and their associated information $D_c^p = \{I_c^p, I_{c,b}^p, \underline{\mathbf{x}}^p, \bar{\mathbf{x}}^p\}$. (values) on p^{th} process as input, it outputs the globally sorted Morton code, and shuffled per-check point information.

The parallel sorting algorithm evenly distributes the sorted Morton IDs among all processes so that the p^{th} process contains list M^p, which is a consecutive part of the sorted global Morton code M, along with corresponding data D_c^p. In addition, the sorting algorithm places identical Morton codes on the same process, which is equivalent to assigning the list of boxes overlapping a grid cell to a single process. This process is responsible for checking intersections of boxes on this grid cell's list.

We find intersecting bounding box pairs on process p by checking all pairs of bounding boxes in $I^p_{c,b}$ which have the same Morton code (line 5–11 of Alg. 4). Line 8 checks the candidate intersecting bounding box pairs to see whether they actually intersect or not. For each pair of intersecting boxes (j, k) we generate two ordered pairs (k, j) and (j, k).

The process p sends the pairs to the process which originally owned the first vesicle in each P^p entry, using a sparse MPI all-to-all communication call. After the MPI communication, the p^{th} process will have a list of intersecting bounding box pairs $P^p = \{(j, k)\}$, where $j \in I^p_b$ and $k \in I^q_b$. These intersecting bounding box pairs are used as the contact candidates for vesicle pairs in STIV calculation below.

3.3.3.2 Phase 2: STIV computation

We compute the STIV between the candidate vesicle pairs found by Alg. 4 to identify pairs that actually intersect.

Algorithm 5 summarizes the steps for computing the STIV. The algorithm starts by defining global index sets I^p for vesicle points stored on each process (this requires an MPI reduction on the set of processes).

Since vesicles are distributed over multiple processes, vesicles may have contact with vesicles on other processes; to compute STIV for all vesicles owned by a process p, positions and velocities of points of vesicles in P^p need to be communicated to p. We refer to these copies of vesicle information as *ghost vesicles*.

Using the contact candidate vesicle pairs P^p, we compute the ghost vesicles' global index sets $\overline{I}^p$ which are used to distribute ghost vesicle point data $\{\overline{X}_0^p, \overline{X}_1^p\}$ (lines 4 and 5). On each process, using the contact detection method Section 3.3.1.3, we compute the contact volumes V^p, the contact volumes' index set I_v^p, the contact volume Jacobian J^p with respect to the local vesicle points on process p, and the contact volume Jacobian $\overline{J}^p$ with respect to the ghost vesicle points copied to this process (line 6).

Initially, the contact volume index set I_v^p is computed from local data and contains local indices. I_v^p is converted to global indices by doing an MPI scan communication on N_v^p.

In addition, process p sends back $\overline{J}^p$ to the owner process of each ghost vesicle and J^p on each process is updated to store rows $J(I_v^p, :)$ of the global Jacobian matrix J.

In the contact volume calculation, each discretization point on the vesicle can only be involved in one contact volume, i.e., each column of J has only a single

Algorithm 5: ContactVolume.

```
   input : Minimum separation distance $d_m$, vesicles' initial position $X_0^p$,
           candidate position $X_1^p$
   output: Contact Volumes $V^p$ and the Jacobian $J^p$
1  Compute $I^p$                                                    // MPIScan
2  $B^p \leftarrow$ getBoundingBox($X_0^p, X_1^p, d_m$)
3  $P^p \leftarrow$ getIntersectingBBPair($B^p, I^p$)                    // Alg. 4
4  $\overline{I}^p \leftarrow$ getGhostVesicleID($P^p$)
5  Send and receive $\{\overline{X}_0^p, \overline{X}_1^p\}$                         // MPIAlltoallSparse
6  $\{V^p, I_v^p, J^p, \overline{J}^p\} \leftarrow$ checkContact($X_0^p, X_1^p, \overline{X}_0^p, \overline{X}_1^p, d_m$)
7  updateContactData($I_v^p, J^p, \overline{J}^p$)    // MPIScan and MPIAlltoallSparse
```

nonzero element. As a consequence, J^p can be compactly stored as vectors g_X^p and g_I^p where g_X^p is the contact volume gradient with respect to X_1^p and g_I^p stores the contact volume indices the contact volume gradient components belong to. Next, we can proceed to solve the LCP and compute the magnitude of the contact force.

3.3.3.3 Solving the linear complementarity problem

The LCP matrix $\mathbf{B} = \mathbf{J}\mathbf{A}^{-1}\mathbf{G}\mathbf{J}^T$ is an $N_v \times N_v$ matrix, where N_v is the number of contact volumes, $N_v = \mathcal{O}(N)$. Each entry $\mathbf{B}_{j,k}$ is the change in the j^{th} contact volume induced by the k^{th} contact force. Due to sparsity in matrices $\mathbf{J}$ and $\mathbf{A}$ (when the locally implicit scheme is used), the matrix $\mathbf{B}$ is sparse and typically diagonally dominant, since most STIV volumes are spatially separate. Two key algorithms are the parallel construction of the LCP matrix $\mathbf{B}$ and applying it to a vector (Alg. 6).

In Alg. 6, we form the LCP matrix $\mathbf{B}$, taking advantage of its sparsity and the sparsity of $\mathbf{J}$. After contact computation (Alg. 5), each process owns a list of contact volumes with indices I_v^p. A process p stores rows $\mathbf{B}(I_v^p,:)$ of $\mathbf{B}$ and $\lambda(I_v^p)$, the local vector of Lagrange multipliers. Note that I_v^p can be empty, which means there is no contact volume on process p. To form $\mathbf{B}(I_v^p,:)$, we loop over all the vesicles residing on process p.

For each contact volume pair (j,k) that a vesicle i is involved in, let g_j and g_k denote the rows j and k of the Jacobian matrix's columns with respect to vesicle i sample point positions, then $\mathbf{B}(j,k)$ is updated as in line 7 of Alg. 6.

Finally, process p needs to send $\mathbf{B}(j,k)$ to process q if $j \in I_v^q$ and $p \neq q$, the owner of j^{th} contact volume.

Since the matrix $\mathbf{B}$ and the vector λ is distributed non-contiguously, some communication is needed to compute the matrix-vector product between $\mathbf{B}$ and λ. Process p needs to send $\lambda(k)$ to process q if there is any non-zero entry $\mathbf{B}(j,k)$ where $j \in I_v^q$ and $k \in I_v^p$. After the communication, we can compute the LCP matrix-vector product $\mathbf{B}\lambda$ locally.

Algorithm 6: Form LCP Matrix.

```
   input  : Contact volume Jacobian $\mathbf{J}$ and CLI scheme matrix $\mathbf{A}$
   output: LCP matrix $\mathbf{B}$
   // For $p^{\text{th}}$ process, form LCP matrix block $\mathbf{B}(I_v^p, :)$.  Use map
      data structure for sparse matrix B
 1 $\mathbf{B} \leftarrow \emptyset$
 2 for each local vesicle $i$ do
 3     for each contact volume $j$ vesicle $i$ involved in do
 4         for each contact volume $k$ vesicle $i$ involved in do
               // accumulate change from $k^{\text{th}}$ contact force to $j^{\text{th}}$
                  contact volume
 5             $\mathbf{B}(j,k) = \mathbf{B}(j,k) + g_j \cdot \mathbf{A}(i,i)^{-1}\mathbf{G}(i,i)g_k$
 6 if $j \notin I_v^p$ then
 7     Send $\mathbf{B}(j,k)$ to process $q$ such that $j \in I_v^q$
```

Algorithm 7: Minimum Map LCP Solver.

```
    require: LCPMatrixApply(), V and $\epsilon$
    output: $\lambda$
  1 $e \leftarrow \epsilon$
  2 $\lambda \leftarrow 0$
  3 while $e > \epsilon$ do
  4     $\mathbf{y} \leftarrow \mathsf{V} +$ LCPMatrixApply($\lambda$)
  5     $\mathbb{A} \leftarrow \{i | y_i < \lambda_i\}$                  // index of active constraints
  6     $\mathbb{F} \leftarrow \{i | y_i \geq \lambda_i\}$
  7     Iteratively solve $\begin{bmatrix} \mathbf{B} & -\mathbf{I} \\ \mathbf{P}_{\mathbb{F}} & \mathbf{P}_{\mathbb{A}} \end{bmatrix} \begin{bmatrix} \Delta\lambda \\ \Delta y \end{bmatrix} = \begin{bmatrix} 0 \\ -\mathbf{P}_{\mathbb{A}}\mathbf{y} - \mathbf{P}_{\mathbb{F}}\lambda \end{bmatrix}$   // $\mathbf{B}$ applied
         by LCPMatrixApply
  8     $\tau \leftarrow$ projectLineSearch($\Delta\lambda$)
  9     $\lambda \leftarrow \lambda + \tau\Delta\lambda$
 10     $e \leftarrow \|\mathbf{H}(\lambda)\|$
```

To solve the LCP, we use the minimum-map Newton method [15, Section 5.8], which only requires the application of the LCP matrix. For the sake of completeness, we briefly summarize the minimum-map Newton method. Let $\mathbf{y} = \mathsf{V} + \mathbf{B}\lambda$. Using the minimum map reformulation, we can convert the LCP to a root-finding problem

$$\mathbf{H}(\lambda) \equiv \begin{bmatrix} h(\lambda_1, y_1) \\ \dots \\ h(\lambda_M, y_M) \end{bmatrix} = 0, \tag{3.3.16}$$

where $h(\lambda_i, y_i) = \min(\lambda_i, y_i)$. This problem is solved by Newton's method (Alg. 7). In the algorithm, $\mathbf{P}_{\mathbb{A}}$ and $\mathbf{P}_{\mathbb{F}}$ are selection matrices: $\mathbf{P}_{\mathbb{A}}\lambda$ selects the rows of λ whose indices are in set $\mathbb{A}$ and zeros out all the other rows. While function $\mathbf{H}$ is not smooth, it is Lipschitz and directionally differentiable, and its B-derivative $\mathbf{P}_{\mathbb{A}}\mathbf{B} + \mathbf{P}_{\mathbb{F}}$ can be formed to find the descent direction for Newton's method [21]. The matrix $\mathbf{P}_{\mathbb{A}}\mathbf{B} + \mathbf{P}_{\mathbb{F}}$ is a sparse matrix, and we use GMRES to solve this linear system. Since $\mathbf{B}$ is sparse and diagonally dominant, in practice the linear system is solved in few GMRES iterations and the Newton solver converges quadratically.

3.4 Results

In this section, we present results characterizing the accuracy, robustness, and efficiency of a locally-implicit time stepping scheme (CLI) combined with our contact resolution framework in comparison to other schemes described in Section 3.3.2: with no contact resolution (i.e., LI scheme) and globally semi-implicit schemes with/without repulsion force (i.e., GI and RGI schemes).

- First, to demonstrate the robustness of our scheme in maintaining the prescribed minimum separation distance with different viscosity contrast ν, we consider two vesicles in an extensional flow, Section 3.4.1.

- In Section 3.4.2, we explore the effect of minimum separation d_m and its effect on collision displacement in shear flow. We demonstrate that the collision scheme has a minimum effect on the shear displacement.

- We present the timing for strong scalability and the weak scalability of our scheme.

- We close this section by reporting the computation cost (wall clock time) for simulations with different volume fractions, and compare the cost with RGI scheme.

Our experiments support the general observation that when vesicles become close, the LI scheme cannot, at a reasonable resolution of discretization, compute the interaction forces between close vesicles [88] and the time stepping becomes unstable. The GI scheme stays stable longer, but the iterative solver requires more and more iterations to reach the desired tolerance, which in turn implies higher computational cost for each time step. Finally, the RGI scheme requires choosing a penalty coefficient, which is typically done on case-by-case basis. If the penalty coefficient is too small, collisions many not be resolved, and an excessively large coefficient increases the error.

3.4.1 Extensional flow

Recall that to maintain accuracy of integral computations at a fixed surface resolution, one needs to ensure that a minimum separation distance is maintained.

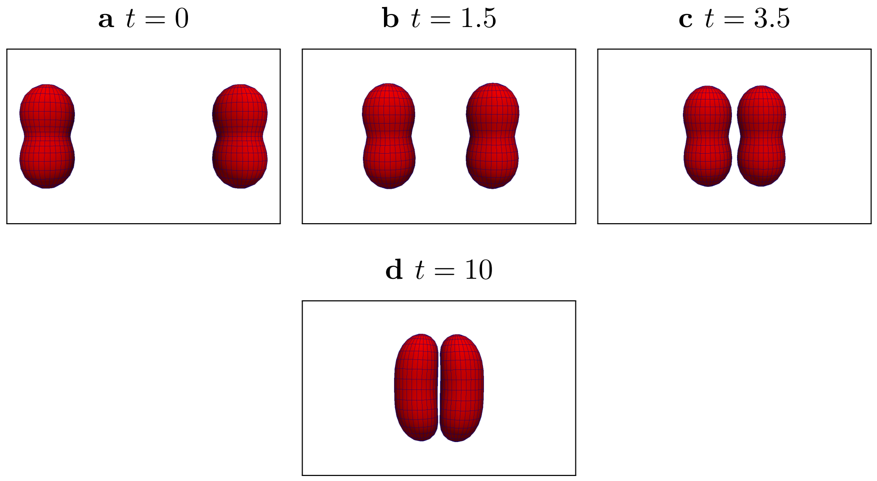
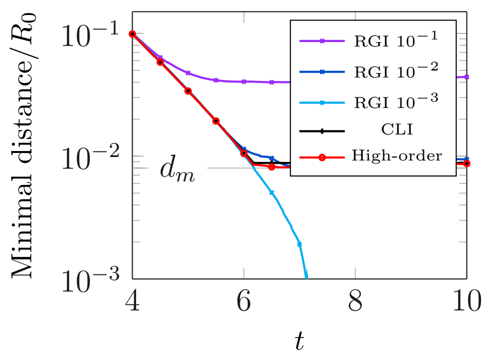


Figure 3.4: SNAPSHOTS OF TWO VESICLES IN EXTENSIONAL FLOW USING THE CLI SCHEME. *As the distance between two vesicles decreases, the CLI scheme maintains the desired minimum separation distance* $d_m = 0.009$ *and two vesicles also maintain a symmetric configuration. The viscosity contrast is 64 in this simulation.*

To demonstrate the ability of our framework to maintain a prescribed separation distance, we consider two vesicles placed symmetrically with respect to the z axis in the extensional flow $\boldsymbol{u} = [-x, y/2, z/2]$. The vesicles have a reduced volume of 0.85 and we use a first-order time stepping with CLI and RGI schemes for the experiments in this test. We run the experiments with different viscosity contrasts and report the resulting minimal distance between vesicles. Snapshots of the vesicle configuration in the CLI scheme are shown in Fig. 3.4.

In Fig. 3.5a, we plot the distance between two vesicles over time using two different schemes (CLI and RGI). The vesicles continue to get closer in the RGI scheme and fluctuates when the repulsion force is present. On the other hand, the CLI scheme consistently maintains the desired minimum separation distance between two vesicles. In Fig. 3.5b, we show the minimum distance between vesicles at the end of simulations, $T = 10$, versus the viscosity contrast with no collision handling. We use adaptive time stepping GI scheme to run the simulations to

a *The distance between two vesicles over time*



b *The minimum distance between two vesicles versus the viscosity contrast*

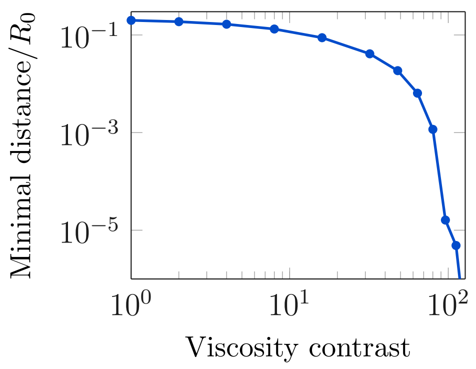


Figure 3.5: Distance between two vesicles in extensional flow. *Fig. 3.5a The distance between two vesicles over time for both CLI and RGI schemes (in all case, the viscosity contrast is* 64*). The vesicles start to be in contact around* $t = 6$*. The black curve shows the minimal distance for the piecewise-linear triangular approximations to a* $p = 32$ *spherical harmonics grid. The red curve shows the estimated minimal distance between high order spectral surfaces (computed on a linear-triangulation to* $4\times$ *upsampled surfaces). For the CLI scheme, the minimum separation distance* d_m *is set to* 0.009*; as discussed in Section 3.3.1.3, we set separation distance to* $(1 + 2\alpha)d_m$ *with* $\alpha = 0.05$ *for* $p = 32$ *piecewise-linear triangular approximation for contact detection. For the RGI scheme, the repulsion coefficient* C_r *is set to* 0.1*,* 0.01 *and* 0.001*.* $R_0 := \sqrt{\text{Area}/4\pi}$ *denotes the effective radius of a vesicle. In these simulations* $R_0 = 1.136$ *and it is used to normalize the minimal distance (the y-axis). The CLI scheme easily maintains the prescribed minimum separation of* d_m*. In this example, the RGI scheme maintains a similar minimal separation distance with* $C_r = 0.01$*, causes large separation distance with* $C_r = 0.1$*; and results in collision with* $C_r = 0.001$*. Fig. 3.5b The final distance (at* $T = 10$*) between two vesicles as viscosity contrast is increased (using GI scheme).*

get the final ($T = 10$) minimum distance between two vesicles. As expected, we observe that the minimum distance between two vesicles decreases as the viscosity contrast is increased. In 3D, however, the minimum distance decreases much faster as the viscosity contrast is increased relative to 2D simulations [53].

To validate our estimates for the error due to piecewise-linear triangular approximation in the minimal separation calculation instead of the exact high-order geometry discretization, we plot the minimum distance at each step for two cases in Fig. 3.5a: (i) the piecewise-linear triangular approximation; and (ii) the corresponding $4\times$ upsampled shape. We observe that, as expected, the actual minimal distance for the smooth, high-order surface is smaller than the minimal distance for piecewise-linear triangular approximation, while the difference between two distances is small compared to the target minimum separation distance.

With the minimum-separation constraint, any desired minimum separation distance between vesicles is maintained and the simulation is more robust as shown in Figs. 3.4 and 3.5. Moreover, the CLI scheme maintains the prescribed minimum separation, while the RGI scheme may fluctuate or collide depending on the prescribed repulsion coefficient C_r. We will show in Section 3.4.5 that with the prescribed repulsion coefficient of $C_r = 0.01$ and similar accuracy compared to the CLI scheme, the RGI does not prevent collisions and is more expensive than the CLI scheme.

3.4.2 Shear flow

We consider vesicles in an unbounded shear flow and explore the effects of minimal separation on shear diffusivity. We report the difference between centroids as a function of the minimum separation distance d_m to demonstrate the effect of the minimum separation constrained system on the dynamics. For this experiment, we set the viscosity contrast to 5.

Snapshot of the flow is shown in Fig. 3.6. In Fig. 3.7, we show the convergence of the vertical displacement between vesicle as a function of the minimum separation d_m and the convergence rate of the scheme.

In Fig. 3.7a, we report the vertical offset between centroids over time as we increase the minimum separation distance. We consider two vesicles of reduced volume 0.85 (to minimize the effect of vesicles' relative orientation on the dynamics) placed in a shear flow with (non-dimensional) shear rate $\chi = 1$. We observed in our previous work that in semi-implicit methods for vesicle suspensions, the stable time step is inversely proportional to shear rate χ [86, Table 6] and [88, Table 4].

With the observation that $\Delta t^{\text{stable}} \propto \chi^{-1}$, we report the results for a single shear

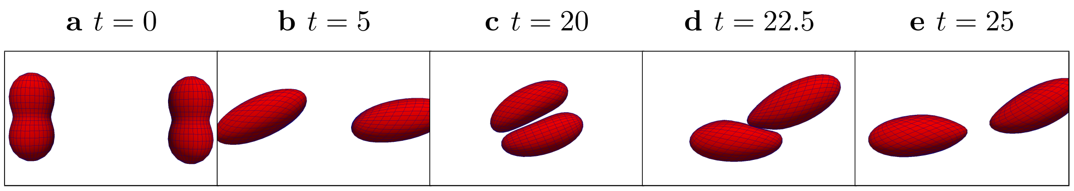
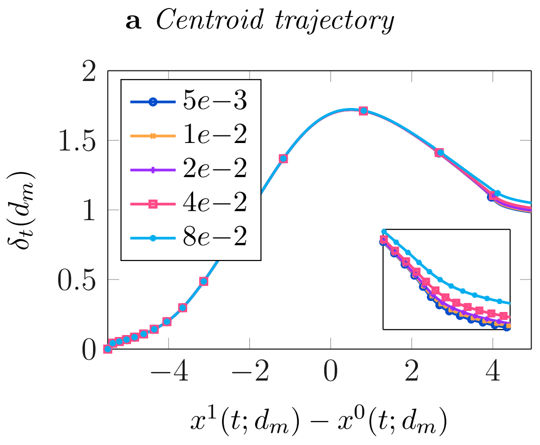


Figure 3.6: Shear flow experiment. *The snapshots of two vesicles in shear flow. Initially, one vesicle is placed at* $[-5.5, 0, 0]$ *and the second vesicle is placed at* $[0, 0, 0]$*. The viscosity contrast for both vesicles is set to* 5*.*



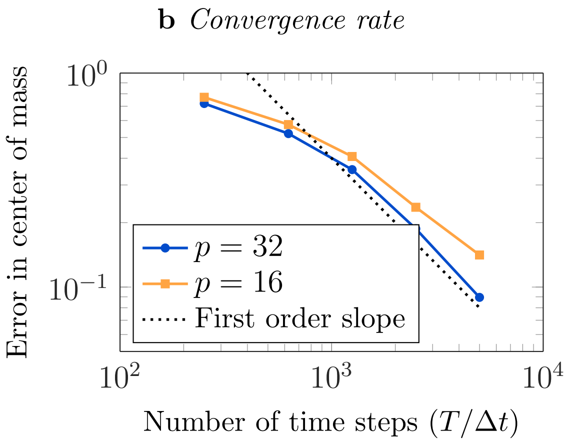


Figure 3.7: The offset $\delta_t(d_m)$ between the centroids of two vesicles in shear flow and convergence rate. *The initial offset is* $\delta_0 = 0$*, viscosity contrast is set to* 5*, and the effective radius is* $R_0 = 1.136$*. Fig. 3.7a This plot shows the vertical offset* $\delta_t(d_m) = |z^1(t) - z^0(t)|$ *over time for different minimum separation distance* d_m*. The* d_m *ranges from* 5e−3 *to* 8e−2*, the simulations converge, as we decrease the* d_m*. Spherical harmonic order* $p = 16$ *is used. Fig. 3.7b This plot shows the error in the final (*$T = 25$*) centroid location as we decrease the time step size for two spherical harmonic orders* $p = 16$ *and* $p = 32$*. We set* $d_m =$ 5e−3 *for* $p = 16$ *and* $d_m =$ 2.5e−3 *for* $p = 32$*. The final error in centroids is calculated with respect to the adaptive GI without repulsion force and* $p = 32$ *and error factor* $E_f = 0.05$*. As expected we observe first order convergence with our CLI scheme.*

rate and the approximate stable time step can be estimated for other rates from this. We let $\delta_t(d_m) := |z^1(t;d_m) - z^0(t;d_m)|$ denote the vertical offset between the centroids of vesicles at time t. Initially, two vesicles are placed with a relative vertical offset $\delta_0 = 0$, as shown in Fig. 3.6. The background shear flow is $\boldsymbol{u} = [\chi z, 0, 0]$ and the viscosity contrast is set to 5 in this experiment. In Fig. 3.7a, we plot δ_t with respect to $x^1(t) - x^0(t)$ for different minimum separation distances. As the minimum separation parameter d_m is decreased, the simulations with minimum-separation constraint converges to the precisely computed trajectory.

In Fig. 3.7b, we report the convergence rate for the final error in centroid locations with some fixed minimum separation distance as we decrease the time step size. We use two $p = 16$ and $p = 32$ and d_m accordingly. We observe first order convergence with CLI scheme. The LI scheme requires very small and often impractical time steps to prevent instability or intersection; we will revisit this in a later section.

3.4.3 Strong scalability

In this section and next, we use report the parallel scaling results for our framework. We used the Stampede1 system at the Texas Advanced Computing Center (TACC) to obtain the strong and weak scalability results. Each compute node in Stampede1 has two eight-core Intel Xeon E5-2680 CPUs running at 2.7GHz and 32GB of memory. We use the periodic Taylor vortex flow, as shown in Fig. 3.8, to

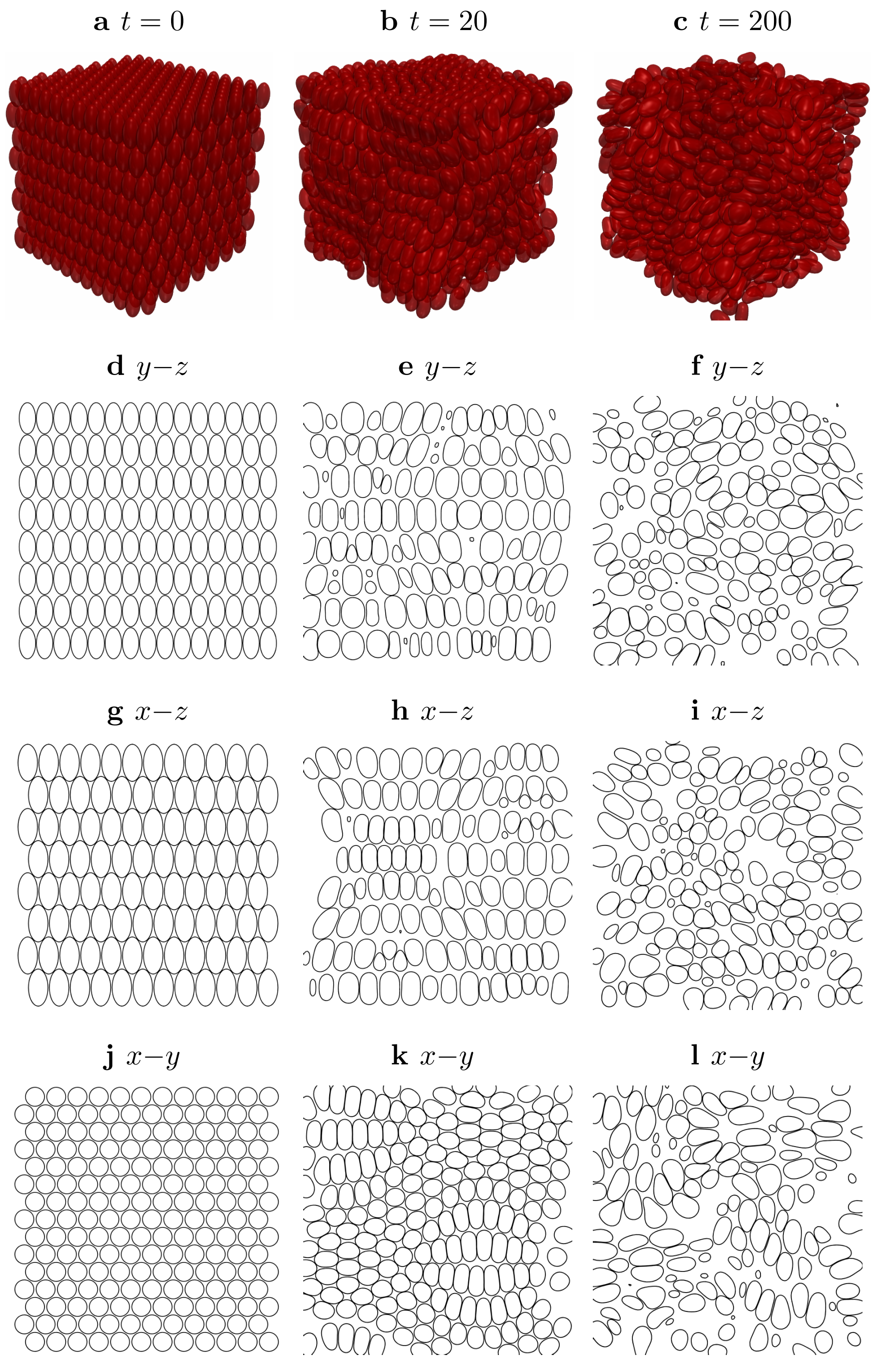


Figure 3.8: Taylor vortex flow experiment. *Snapshots of 1440 vesicles in Taylor vortex flow.*

investigate strong scaling. For this flow, the background velocity is

$$\begin{aligned} \boldsymbol{u}^\infty(x,y,z) = \alpha \sin\left(\frac{2\pi x}{L}\right)\cos\left(\frac{2\pi y}{L}\right)\sin\left(\frac{2\pi z}{L}\right)\boldsymbol{e}_1 \\ + \alpha \cos\left(\frac{2\pi x}{L}\right)\sin\left(\frac{2\pi y}{L}\right)\sin\left(\frac{2\pi z}{L}\right)\boldsymbol{e}_2, \end{aligned} \tag{3.4.1}$$

where L is the periodic length and α is the scaling factor. For the simulations in this section, we choose $L = 28.4$ and $\alpha = 1$. Resulting timings are shown in Fig. 3.9.

We used the following simulation parameters:

- The number of vesicles is 1440; the vesicles are ellipsoidal, of effective radius $R_0 = 1.34$ with reduced volume 0.91 and the bending modulus is 0.1.
- Vesicle volume fraction is 58%.
- For spatial discretization, order $p = 16$ spherical harmonic were used, and the grid was upsampled to twice the resolution for collision detection.
- The time horizon is $T = 2$ and the time step size is $\Delta t = 0.1$.
- The block-diagonal solver relative tolerance is chosen to be $1e{-}5$.

The average number of contacts per vesicle stays about 2 per time step in our simulation. In Fig. 3.9, we report the total CPU time (wall-clock-time $\times$ CPU cores) for the number of cores ranging from 16 (1 compute node) to 960 (60 compute nodes). We achieve a speedup of 20.1 for the wall-clock-time or 33.5% strong scaling efficiency.

Figure 3.9 also shows the breakdown of the time spent in different parts of the code. As is evident in Fig. 3.9b, there are two main regimes in the scaling. When

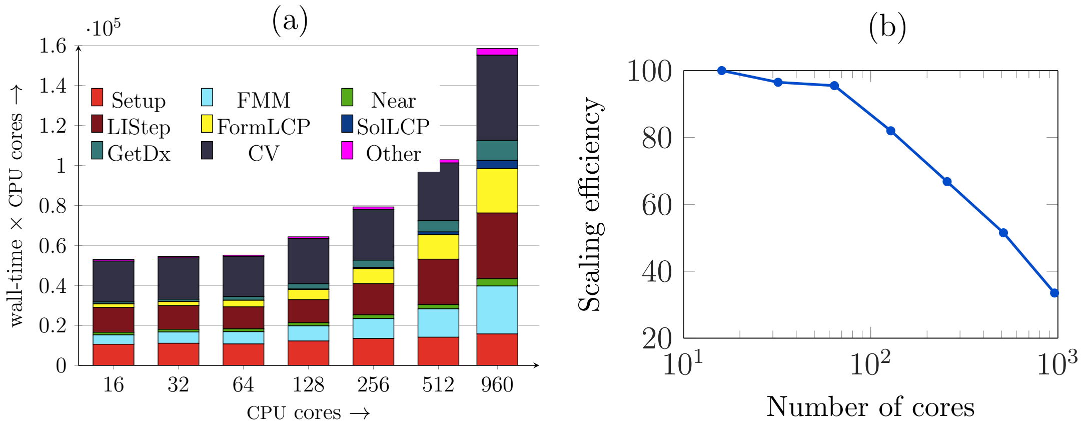


Figure 3.9: Strong scalability for periodic Taylor-vortex flow. *Setup is the setup phase for FMM, FMM is the actual FMM evaluation, Near is the near-singular calculation, (FMM+Near is line* 2 *of Alg. 3), LIStep represents the solve phase of LI scheme FormLCP and SolLCP are lines 8 and 9 respectively in Alg. 3 and constitute the time spent on LCP, CV is the contact volume calculation which corresponds to Alg. 5, GetDx is the phase of calculating new candidate solution (line* 12 *of Alg. 3), and Other represents all other calculations.*

there are few vesicles per core (going from 64 to 128 cores), the load imbalance for the collision becomes more pronounced. The fraction of time spent on contact volume computation (CV) grows due to load imbalance with respect to the number of collisions and STIV calculation. Simple rebalancing by re-assigning different numbers of vesicles to processors is ineffective because of the subsequent increase in the load imbalance for the solver and FMM portions. The computational load of collision computations is dynamic and depends on the flow regime. To improve scalability of the collision computation in strong scaling regime, careful dynamic rebalancing, that takes into account the trade off between collision and solver balancing, will be needed.

In our experiments, we observe that when the grain size (i.e., the number of vesicles per processor) is not too small, randomly distributing vesicles among processors with equal number of vesicles per processor achieves a good load balance

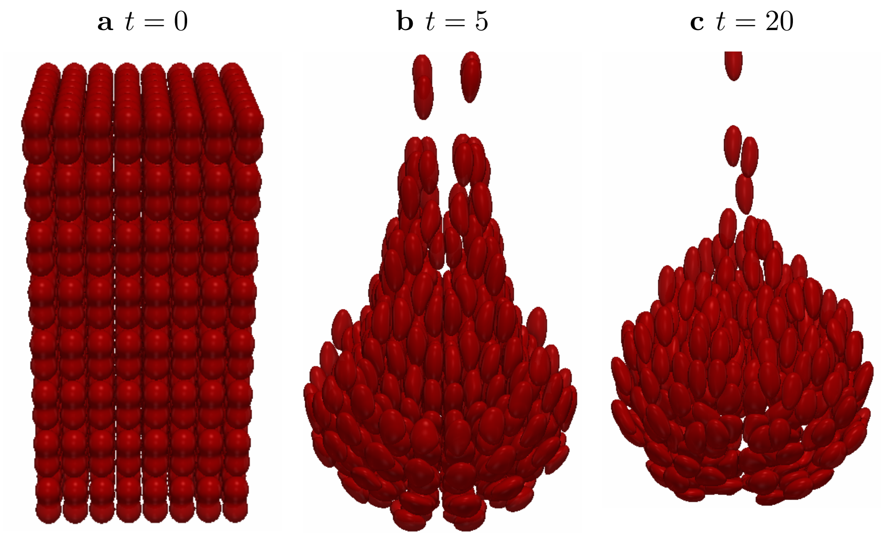


Figure 3.10: SEDIMENTATION OF POLY-DISPERSE VESICLES. *The snapshots of 512 (*$8 \times 8 \times 8$ *lattice) vesicles sedimenting under gravity. Each vesicle has a reduced volume of* 0.91*, bending modulus is in the range* $[0.05, 0.1]$*, viscosity contrast in the range* $[0.5, 5]$ *and an excess density of* 1*. We use time step size* $\Delta t = 0.01$ *and time horizon* $T = 30$ *in this simulation, the initial lattice has volume fraction of* 53%*.*

for both solver and collision parts of the code.

The solver time (LIStep) dominates for very small grain size, since the linear system is block-diagonal and each process requires a different number of GMRES iterations for convergence; processes with fewer GMRES iterations will wait for processes requiring more GMRES iterations.

3.4.4 Weak Scalability

To showcase different flows, we use the sedimentation of a poly-disperse suspension of vesicles as in Fig. 3.10 on 16K CPU cores for our weak scaling study. The scaling results are shown in Fig. 3.11. We use time step size of $\Delta t = 0.01$ and a time horizon $T = 0.1$. All other simulation parameters are equal to those of the strong scaling test.

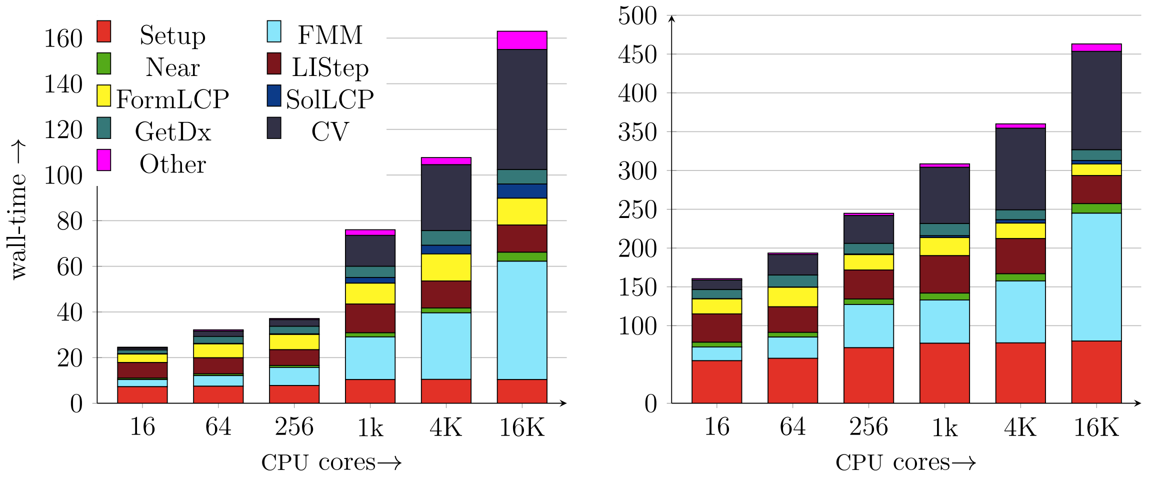


Figure 3.11: Weak scalability results for poly-disperse sedimentation. *The left figure is with grain size* 1 *vesicle per core, the right figure is with grain size* 8 *vesicles per core. The flow snapshots are shown in Fig. 3.10 and the flow parameters are outlined therein. For each case, we present a breakdown of the wall-clock-time spent on each of the different functions in our algorithm. See the caption of Fig. 3.9 for the description of the labels.*

We present two sets of results for 1 vesicle per core (Fig. 3.11 left) and 8 vesicles per core (Fig. 3.11 right). We present a breakdown of the time spent on different functions of our algorithm as we scale from 16 cores to 16K cores. Similar to the strong scaling case, load imbalance with respect to the number of collisions causes the timing to grow. In this flow, different regions of space have very different number of collisions.

Another factor that affects the timing is the number of contact-resolving iterations, which grows from 4 to 8 as we increase the number of cores from 16 to 16K (i.e., as the problem size increases). For uniform lattice as in sedimentation experiment Fig. 3.10, the number of contact-resolving iterations stabilizes to about 8 as we increase the number of cores.

3.4.5 High volume-fraction flows

In our final experiment, we investigate the effectiveness of our scheme in modeling flows with high volume-fraction ϕ. We use the wall-clock-time for a fixed time horizon to quantify the cost of simulation for each scheme (CLI, RGI, and time-adaptive RGI).

For this experiment, we use Taylor-vortex flow with 168 prolate vesicles distributed on a staggered lattice as shown in Fig. 3.12. The shape and distribution of vesicles are chosen to achieve high volume fractions), the simulation time horizon is set to $T = 15$. We change the periodic length L and spacing between vesicles to obtain different volume fractions. All of the simulations are executed on a dedicated node with the same type of CPU with 1 MPI process to ensure the wall time used is calculated consistently.

We run three sets of experiments:

1. *CLI scheme:* We use the CLI scheme to run the simulations with different volume fractions ϕ and different time step sizes Δt. The minimum separa-

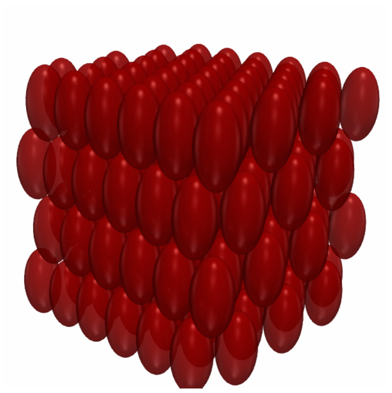


Figure 3.12: VOLUME FRACTION EXPERIMENT. *An example of the initial distribution of 168 vesicles in Taylor-vortex flow. We modify the spacing between vesicles to obtain different volume fractions.*

tion d_m is set to 0.009. We report the total wall-clock-time (in seconds) in Table 3.4a. Since the time stepping scheme is locally implicit, large time steps cause the simulation to diverge. We mark those cases by "LI-div". As

ϕ	Δt			
	0.1	0.2	0.4	0.8
0.35	3.1e3	2.1e3	1.8e3	LI-div
0.40	3.3e3	2.2e3	1.9e3	LI-div
0.45	3.7e3	2.5e3	2.1e3	LI-div
0.50	4.2e3	2.7e3	2.3e3	LI-div
0.55	5.2e3	3.4e3	2.7e3	LI-div

(a) *CLI scheme*

ϕ	E_f				
	0.25	0.5	1.0	2.0	4.0
0.35	150e3	150e3	142e3	136e3	Col
0.40	139e3	135e3	133e3	127e3	Col
0.45	Limit	172e3	173e3	173e3	Col
0.50	Limit	158e3	158e3	154e3	Col
0.55	Limit	143e3	139e3	Col	Col

(b) *Adaptive RGI scheme*

Table 3.4: WALL-CLOCK-TIME VS. VOLUME FRACTION. *Wall-clock-time (seconds) for simulating* 168 *vesicles in Taylor vortex flow with different volume fractions. Table 3.4a For the CLI scheme,* $d_m = 0.009$. *Table 3.4b For the RGI scheme, the* $C_r = 0.01$.

C_r	Δt					
	0.025	0.05	0.1	0.2	0.4	0.8
0.01	49.7e3	Col	Col	Col	Col	RGI-div
0.02	46.6e3	33.3e3	20.2e3	Col	Col	RGI-Div
0.05	42.7e3	30.1e3	18.8e3	11.1e3	Col	RGI-Div
0.10	40.1e3	29.3e3	17.5e3	10.5e3	Col	RGI-Div

Table 3.5: WALL-CLOCK-TIME VS. REPULSION COEFFICIENT. *Wall-clock-time (seconds) for the non-adaptive RGI scheme with different repulsion coefficient* C_r *and fixed time step* Δt. *The volume fraction is fixed at* $\phi = 0.5$.

expected, the wall-clock-time shows weak dependence on the volume fraction since there are more collisions.

2. *Adaptive time stepping with repulsion:* We use adaptive time stepping RGI scheme to run simulations with different error factors E_f, different volume fractions ϕ, and a fixed repulsion coefficient $C_r = 0.01$, which maintains similar separation distance as $d_m = 0.009$ for the CLI scheme above, Fig. 3.5a. In the adaptive scheme, the error factor is the tolerance for the error committed in each simulation time unit [58]. If the problem is non-stiff, one expects the time step to be proportional to E_f. Therefore, the counterpart of E_f in non-adaptive schemes is the step size Δt. In Table 3.4b, we report the wall-clock time in seconds provided that the simulation finishes. If Δt of the adaptive scheme is reduced below $(1e{-}11)$ we abort the simulation (these are marked as “Limit” in the table). If a collision occurs, we stop the simulation and mark it as “Col”.

   In Table 3.4b, we chose the widest possible range for E_f (going from .25 to 4.0) to present the full picture with respect to the simulation cost. Comparing the results in Table 3.4, we see that the adaptive scheme is one to two orders of magnitude more expensive for similar cases. For example, for $\phi = 0.55$, the RGI scheme with $E_f = 0.5$ requires $143e3$ seconds compared to $2.7e3$ of the CLI scheme with $\Delta t = 0.4$; a $53\times$ speedup.

3. *Non-adaptive RGI scheme:* To compare the cost of RGI scheme with that of CLI, we report the wall-clock time for fixed Δt and different repulsion coefficients C_r; the volume fraction for this experiment is fixed at $\phi = 0.5$. The time steps are chosen to include those used for the CLI scheme in Table 3.4.

In Table 3.5 we report the wall-clock time (in seconds) when the simulation finishes. The cases where the scheme diverges, because the chosen time step is unstable due to the stiffness introduced by the repulsion force, are marked as “RGI-div”.

Larger repulsion coefficients avoid collision at the expense of accuracy. In [58, Figure 5(f)], the error analysis for different repulsion coefficients shows that large repulsion coefficient will introduce significant error in the center of mass trajectory. The largest repulsion coefficient we test here is 0.10 which already has a significant error. To match the error of CLI, C_r needs to be set around 0.01 that in turn requires very small time step.

The results in Table 3.5 show that for a fixed Δt, the repulsion coefficient needs to be adjusted for the simulation to succeed. There are many factors influencing the choice of repulsion coefficient, e.g., bending modulus, viscosity contrast, volume fraction, reduced volume, background flow and vesicle shape, which makes an automatic choice difficult.

Comparing to the row corresponding to volume fraction $\phi = 0.5$ of the CLI scheme in Table 3.4, the CLI scheme is at least $5\times$ faster than RGI with the “right” choice of the repulsion coefficient.

3.5 Conclusion

We have introduced new parallel algorithms for efficient 3D simulation of non-dilute suspensions of deformable particles immersed in Stokesian fluid in this chapter. We demonstrated the parallel scaling of the algorithms on up to 16K CPU cores. Moreover, we demonstrated through numerical experiments that our scheme is orders of magnitude faster than the alternatives for several setups.

Chapter 4

Parallel algorithms for direct simulation of blood flow in complex geometry

4.1 Introduction

The ability to simulate complex biological flows from first principles has the potential to provide insight into complicated physiological processes. Simulation of blood flow, in particular, is of paramount biological and clinical importance. Blood vessel constriction and dilation affects blood pressure, forces between RBCs can cause clotting, various cells migrate differently through microfluidic devices.

Achieving accurate, robust and scalable simulation for a blood flow requires that the system meets a number of stringent requirements. While previous work has made significant progress [58, 87], we focus on several new infrastructure components essential for handling confined flows and arbitrarily long-time, high volume fractions RBC flows; in particular, our work is able to realize each of these goals.

We formulate the viscous flow in blood vessels as an integro-differential equation and make use of fast scalable summation algorithms for efficient implementation, as in prior RBC simulations [105]. This is the only approach to date that maintains high accuracy at the microscopic level while avoiding expensive discretization of fluid volume: all degrees of freedom reside on the surfaces of RBCs and blood vessels.

To achieve high accuracy with minimal degrees of freedom per cell, we required a smooth yet compact boundary representation; we use spherical harmonic representations for cell boundaries and high-order polynomials for the blood vessels. We update RBC positions with a semi-implicit time stepping scheme.

The most important novel aspects of our system include: (a) handling the RBC-blood vessel interaction with a fully parallel, high-order boundary integral equation solver; (b) explicit handling of collisions with a parallel constraint-based resolution and detection algorithm. The former is essential for modeling confined flows, while the latter is essential for handling high-volume fraction flows at long time scales without excessively small time steps or fine spatial discretizations.

Our contributions

1. We present a parallel platform for long-time simulations of RBCs through complex blood vessels. The extension to suspensions of various particulates (fibers, rigid bodies etc.) is straightforward from the boundary integral formulation. Flows through several complicated geometries are demonstrated.

2. We have parallelized a boundary solver for elliptic PDEs on smooth complex geometries in 3D. By leveraging the parallel fast-multipole method of [56] and

the parallel forest of quadtrees of [13], we are able to achieve good parallel performance and load balancing.

3. We have extended the parallel collision handling of [54] to include rigid 3D boundaries composed of patches.

4. We present weak and strong scalability results of our simulation on the Skylake cluster and weak scaling results on the Knights Landing cluster on Stampede2 at the Texas Advanced Computing Center along with several visualizations of long-time, large-scale blood cell flows through vessels. We observe 49% strong scaling efficiency for a 32-fold increase of compute cores. In our largest test on 12288 cores, we simulate 1,048,576 RBCs in a blood vessel composed of 2,097,152 patches with weak scaling efficiency of 71% compared to 192 cores (Fig. 4.7). In each time step, this test uses over three billion degrees of freedom and over four billion surface elements (triangles).

5. We are able to simulate realistic human blood flows with RBC volume fractions over 47% (Fig. 4.4).

Limitations Despite the advantages and contributions of the computational framework presented here, our work has some limitations. We have made several simplifications in our model for RBCs. We are restricted to the low Reynolds number regime, i.e., small arteries and capillaries. We use a simplified model for RBCs, assuming the cell membranes to be inextensible and with no in-plane shear rigidity, In our scheme, each RBC is discretized with an equal number of points, despite the varied behavior of the velocity through the vessel. Adaptive refinement is required in order to resolve the velocity accurately. Finally, the blood vessel is constructed to satisfy certain geometric constraints that allow for the solution of Eq. (4.2.5) via singular integration. This can be overcome through uniform refinement, but a parallel adaptive algorithm is required to maintain good performance.

Related work: blood flow Large-scale simulation of RBC flows typically fall into four categories: (a) *Immersed boundary (IB)* and *immersed interface methods*; (b) particle-based methods such as *lattice Boltzmann* (LB), *dissipative particle dynamics (DPD)* and *smoothed particle hydrodynamics (SPH)* (c) multiscale network-based appraoches and (d) *boundary integral equation (BIE)* approaches. For a comprehensive review of general blood flow simulation methods, see [26]. IB methods can produce high-quality simulations of heterogeneous particulate flows in complex blood vessels [6, 7, 111]. These methods typically require a finite element solve for each RBC to compute membrane tensions and use IB to couple the stresses with the fluid. This approach quickly becomes costly, especially for high-order elements, and although reasonably large simulation have been achieved, large-scale parallelization has remained a challenge. A different approach to simulating blood flow is with multiscale reduced-order models. By making simplifying assumptions about the fluid behavior throughout the domain and transforming the complex fluid system into a simpler flow problem, the macroscopic behaviors of enormous capillary systems can be characterized [72, 73] and scaled up to thousands of cores [71]. This comes at a cost of local accuracy; by simulating the flows directly, we are able to accurately resolve local RBC dynamics that are not captured by such schemes.

Particle-based methods have had the greatest degree of success at large-scale blood flow simulations [29, 32, 90, 91]. These types of approaches are extremely flexible in modeling the fluid and immersed particles, but usually suffer from numerical stiffness that requires very small time steps and are computationally demanding, for a given target accuracy. For a comprehensive review, see [115]. There have also been recent advances in coupling a particle-based DPD-like scheme with IB in parallel [114, 116], but the number of RBCs simulated and the complexity of the boundary seems to be limited.

BIE methods have successfully realized large-scale simulations of millions of RBCs [87] in free space. Recently, new methods for robust handling of collisions between RBCs in high-volume fraction simulations have been introduced [54, 58]. This approach is versatile and efficient due to only requiring discretization of RBCs and blood vessel surfaces, while achieving high-order convergence and optimal complexity implementation due to fast summation methods [44, 88, 95, 96, 104, 105, 122]. To solve elliptic partial differential equations, BIE approaches have been successful in several application domains [12, 106, 107, 117]. However, to our knowledge, there has been no work combining a Stokes boundary solver on arbitrary complex geometries in 3D with a collision detection and resolution scheme to simulate RBC flows at large scale. This work aims to fill this gap, illustrating that this can be achieved in a scalable manner.

Related work: collisions Parallel collision detection methods are a well-studied area in computer graphics for both shared memory and GPU parallelism [41, 50, 61]. [17, 38] detect collisions between rigid bodies in a distributed memory architecture via domain decomposition. [70] constructs a spatial hash to cull collision candidates and explicitly check candidates that hash to the same value. The parallel geometry and physics-based collision resolution scheme detailed in [112] is most similar to the scheme used in this work. However, such discrete collision detection schemes require small time steps to guarantee detections which can become costly for high-volume fraction simulations.

4.2 Formulation and solver overview

4.2.1 Problem summary

We simulate the flow of N cells with deformable boundary surfaces γ_i, $i = 1, \ldots, N$ in a viscous Newtonian fluid in a domain $\Omega \subset \mathbb{R}^3$ with a fixed boundary Γ. The governing partial differential equations (PDEs) describing the conservation of momentum and mass are the incompressible Stokes equations for the velocity $\boldsymbol{u}$ and pressure p, combined with velocity boundary conditions on Γ. Additionally, we model cell membranes as massless, so the velocity $\mathbf{X}_t$ of the points on the cell surface coincides with the flow velocity:

$$-\mu\Delta\boldsymbol{u}(\boldsymbol{x}) + \nabla p(\boldsymbol{x}) = \mathbf{F}(\boldsymbol{x}) \quad \text{and} \quad \nabla\cdot\boldsymbol{u}(\boldsymbol{x}) = 0, \quad \boldsymbol{x}\in\Omega, \tag{4.2.1}$$

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{x}), \quad \boldsymbol{x}\in\Gamma, \tag{4.2.2}$$

$$\mathbf{X}_t = \boldsymbol{u}(\mathbf{X}), \quad \boldsymbol{X}\in\gamma_i(t), \tag{4.2.3}$$

where μ is the viscosity of the ambient fluid; in our simulations, we use a simplified model with the viscosity of the fluid inside the cells also being μ although our code supports arbitrary viscosity contrast. The right-hand side force in the momentum equation is due to the sum of tension and bending forces $\mathbf{f} = \mathbf{f}_\sigma + \mathbf{f}_b$; it is concentrated on the cell surfaces. We assume that cell surfaces are inextensible, with bending forces determined by the Canham-Helfrich model [14, 37], based on the surface curvature, and surface tension determined by the surface incompressibility condition $\nabla_{\gamma_i} \cdot \boldsymbol{u} = 0$ resulting in

$$\mathbf{F}(\boldsymbol{x}) = \sum_i \int_{\gamma_i} \mathbf{f}(\boldsymbol{y})\delta(\boldsymbol{x} - \boldsymbol{y})d\boldsymbol{y}$$

(see, e.g., [88] for the expressions for $\mathbf{f}$). Except on inflow and outflow regions of the vascular network, the boundary condition $\boldsymbol{g}$ is zero, modeling no-slip boundary condition on blood vessel walls.

4.2.1.1 Boundary integral formulation

To enforce the boundary conditions on Γ, we use the standard approach of computing $\boldsymbol{u}$ as the sum of the solution $\boldsymbol{u}^{\text{fr}}$ of the free-space equation Eq. (4.2.1) without boundary conditions but with non-zero right-hand side $\mathbf{F}(\boldsymbol{x})$, and the second term $\boldsymbol{u}^{\Gamma}$ obtained by solving the homogeneous equation with boundary conditions on Γ given by $\boldsymbol{g} - \boldsymbol{u}^{\text{fr}}$.

Following the approach of [54, 64, 75, 77], we reformulate Eqs. (4.2.1) and (4.2.2) in the integral form. The free-space solution $\boldsymbol{u}^{\text{fr}}$ can be written directly as the sum of the single-layer Stokes potentials $\boldsymbol{u}^{\gamma_i}$:

$$\boldsymbol{u}^{\gamma_i}(\boldsymbol{x}) = (S_i\mathbf{f})(\boldsymbol{x}) = \int_{\gamma_i} S(\boldsymbol{x}, \boldsymbol{y})\mathbf{f}(\boldsymbol{y})d\boldsymbol{y}, \quad \boldsymbol{x} \in \Omega. \tag{4.2.4}$$

To obtain $\boldsymbol{u}^{\Gamma}$, we reformulate the homogeneous volumetric PDE with nonzero boundary conditions as a boundary integral equation for an unknown double-layer density ϕ defined on the domain boundary Γ:

$$\left(\frac{1}{2}I + D + N\right)\phi = \tilde{D}_{\Gamma}\phi = \boldsymbol{g} - \boldsymbol{u}^{\text{fr}}, \quad \boldsymbol{x} \in \Gamma, \tag{4.2.5}$$

where the double-layer operator is $D\phi(\boldsymbol{x}) = \int_\Gamma D(\boldsymbol{x},\boldsymbol{y})\phi(\boldsymbol{y})d\boldsymbol{y}$ with double-layer Stokes kernel $D(\boldsymbol{x},\boldsymbol{y})$, and the null-space operator needed to make the equations full-rank is defined as $(N\phi)(\boldsymbol{x}) = \int_\Gamma(\boldsymbol{n}(\boldsymbol{x})\cdot\phi(\boldsymbol{y}))\boldsymbol{n}(\boldsymbol{y})d\boldsymbol{y}$ (cf. [53]). One of the key differences between this work and previous free-space large-scale simulations is the need to solve this equation in a scalable way. Once the density ϕ is computed, the velocity correction $\boldsymbol{u}^\Gamma$ is evaluated directly as $\boldsymbol{u}^\Gamma = D\phi$.

The equation for the total velocity $\boldsymbol{u}(\boldsymbol{x})$ at any point $\boldsymbol{x} \in \Omega$ is then given by

$$\boldsymbol{u} = \boldsymbol{u}^{\text{fr}} + \boldsymbol{u}^\Gamma = \sum_{i=1}^{N} \boldsymbol{u}^{\gamma_i} + \boldsymbol{u}^\Gamma. \tag{4.2.6}$$

In particular, this determines the update equation for the boundary points of cells; see Eq. (4.2.3).

Contact formulation In theory, the contacts between surfaces are prevented by the increasing fluid forces as surfaces approach each other closely. However, ensuring accuracy of resolving forces may require prohibitively fine sampling of surfaces and very small time steps, making large-scale simulations in space and time impractical. At the same time, as shown in [53], interpenetration of surfaces results in a catastrophic loss of accuracy due to singularities in the integrals.

To guarantee that our discretized cells remain interference-free, we augment Eqs. (4.2.1) and (4.2.2) with an explicit inequality constraint preventing collisions. We define a vector function $V(t)$ with components becoming strictly negative if any cell surfaces intersect each other, or intersect with the vessel boundaries Γ. More specifically, we use the *space-time interference volumes* introduced in [36] and applied to 3D cell flows in [54]. Each component of V corresponds to a single connected overlap. The interference-free constraint at time t is then simply

$V(t) \geq 0$.

For this constraint to be satisfied, the forces $\mathbf{f}$ are augmented by an artificial collision force, i.e., $\mathbf{f} = \mathbf{f}_b + \mathbf{f}_\sigma + \mathbf{f}_c$, $\mathbf{f}_c = \nabla_u V^T \lambda$, where λ is the vector of Lagrange multipliers, which is determined by the additional *complementarity* conditions:

$$\lambda(t) \geq 0, \quad V(t) \geq 0, \quad \lambda(t) \cdot V(t) = 0, \tag{4.2.7}$$

at time t, where all inequalities are to be understood component-wise.

To summarize, the system that we solve at every time step can be formulated as follows, where we separate equations for different cells and global and local parts of the right-hand side, as it is important for our time discretization:

$$\boldsymbol{X}_t = \left(\sum_{j \neq i} S_j \mathbf{f}_j + D\phi \right) + S_i \mathbf{f}_i, \quad \text{for points on } \gamma_i, \tag{4.2.8}$$

$$\nabla_{\gamma_i} \cdot \boldsymbol{X}_t = 0, \quad \mathbf{f}_j = \mathbf{f}(\boldsymbol{X}_j, \sigma_j, \lambda), \tag{4.2.9}$$

$$B_\Gamma \phi = \boldsymbol{g} - \sum_j S_j \mathbf{f}_j, \quad \text{for points on } \Gamma, \tag{4.2.10}$$

$$\lambda(t) \geq 0, \quad V(t) \geq 0, \quad \lambda(t) \cdot V(t) = 0. \tag{4.2.11}$$

At every time step, (4.2.11) results in coupling of all close γ_i's, which requires a non-local computation. We follow the approach detailed in [53, 54] to define and solve the *nonlinear complementarity problem* (NCP) arising from cell-cell interactions in parallel, and extend it to prevent intersection of cells with the domain boundary Γ, as detailed in Section 4.4.

4.2.2 Algorithm Overview

Next, we summarize the algorithmic steps used to solve the constrained integral equations needed to compute cell surface positions and fluid velocities at each time step. In the subsequent sections, we detail the parallel algorithms we developed to obtain good weak and strong scalability, as shown in Section 4.5.

Overall Discretization. RBC surfaces are discretized using a spherical harmonic representation, with surfaces sampled uniformly in the standard latitude-longitude sphere parametrization. The blood vessel surfaces Γ are discretized using a collection of high-order tensor-product polynomial patches, each sampled at Clenshaw-Curtis quadrature points. The space-time interference volume function $V(t)$ is computed using a piecewise-linear approximation as described in [54]. For time discretization, we use a locally-implicit first order time-stepping (higher-order time stepping can be easily incorporated). Interactions between RBCs and the blood vessel surfaces are computed *explicitly*, while the self-interaction of a single RBC is computed *implicitly*.

The state of the system at every time step is given by a triple of distributed vectors $(\boldsymbol{X}, \sigma, \lambda)$. The first two (cell surface positions and tensions) are defined at the discretization points of cells. The vector λ has variable length and corresponds to connected components of collision volumes. We use the subscript i to denote the subvectors corresponding to i-the cell. $\boldsymbol{X}$ and σ are solved for as a single system, including the incompressibility constraint Eq. (4.2.9). To simplify exposition, we omit σ in our algorithm summary, which corresponds to dropping $\mathbf{f}_\sigma$ in the Stokes equation, and dropping the surface incompressibility constraint equation.

Algorithm summary At each step t, we compute the new positions $\boldsymbol{X}_i^+$ and collision Lagrange multipliers λ^+ at time $t^+ = t + \Delta t$. We assume that in the initial configuration there are no collisions, so the Lagrange multiplier vector λ is zero. Discretizing in time, Eq. (4.2.8) becomes

$$\boldsymbol{X}_i^+ = \boldsymbol{X}_i + \Delta t \left(\sum_{j \neq i} S_j \mathbf{f}_j(\boldsymbol{X}_j, \lambda) + D\phi(\boldsymbol{X}_j, \lambda) \right) + \Delta t S_i \mathbf{f}_i(\boldsymbol{X}_i^+, \lambda^+).$$

At each single time step, we perform the following steps to obtain $(\boldsymbol{X}^+, \lambda^+)$ from $(\boldsymbol{X}, \lambda)$. Below evaluation of integrals implies using appropriate (smooth, near-singular or singular) quadrature rules on cell or blood vessel surfaces.

1. Compute the explicit part $\boldsymbol{b}$ of the position update (first term in Eq. (4.2.8)).

   (a) Evaluate $\boldsymbol{u}^{\text{fr}}$ from $(\boldsymbol{X}, \lambda)$ on Γ with Eq. (4.2.4).

   (b) Solve Eq. (4.2.5) for the unknown density ϕ on Γ using GMRES.

   (c) For each cell, evaluate $\boldsymbol{u}_i^\Gamma = D\phi$ at all cell points $\boldsymbol{X}_i$.

   (d) For each cell i, compute the contributions of other cells to $\boldsymbol{X}_i^+$: $\boldsymbol{b}_i^c = \boldsymbol{u}^{\text{fr}} - u^{\gamma_i} = \sum_{j \neq i} S_j \mathbf{f}_j$.

   (e) Set $\boldsymbol{b}_i = \boldsymbol{u}_i^\Gamma + \boldsymbol{b}_i^c$.

2. Perform the implicit part of the update: solve the NCP obtained by treating the second (self-interaction) term in Eq. (4.2.8) while enforcing the complementarity constraints Eq. (4.2.7), i.e., solve

$$\boldsymbol{X}_i^+ = \boldsymbol{X}_i + \Delta t(\boldsymbol{b}_i + S_i \boldsymbol{f}_i(\boldsymbol{X}_i^+, \lambda^+)), \tag{4.2.12}$$

$$\lambda(t^+) \geq 0, \quad V(t^+) \geq 0, \quad \lambda(t^+) \cdot V(t^+) = 0. \tag{4.2.13}$$

Items 1a to 1d all require evaluation of global integrals, evaluated as sums over quadrature points; we compute these sums in parallel with PVFMM. In particular, Item 1b uses PVFMM as a part of each matrix-vector product in the GMRES iteration. These matrix-vector product, as well as Items 1a, 1c and 1d require near-singular integration to compute the velocity accurately near RBC and blood vessel surfaces; this requires parallel communication to find non-local evaluation points. Details of these computations are discussed in Section 4.3.

The NCP problem is solved using a sequence of *linear complementarity problems* (LCPs). Algorithmically, this requires parallel searches of collision candidate pairs and the repeated application of the distributed LCP matrix to distributed vectors. Details of these computations are provided in Section 4.4.

4.3 Boundary Solver

A main challenge in incorporating prescribed flow boundary conditions $\boldsymbol{g}$ on the domain boundary Γ is the approximation and solution of the boundary integral problem Eq. (4.2.5). Upon spatial discretization, this is an extremely large, dense linear system that must be solved at every time step due to the changing free space solution $\boldsymbol{u}^{\text{fr}}$ on the right hand side. Since we aim at a scalable implementation, we do not assemble the operator on the left hand side but only implement the corresponding matvec-operation, i.e., its application to vectors. Combined with an iterative solver (we use GMRES), this matvec operation is sufficient to solve Eq. (4.2.5). Application of the double-layer operator D to vectors amounts to a near-singular quadrature for points close to Γ. Controlling the error in this computation requires a tailored quadrature scheme. This scheme is detailed below, where we put a particular emphasis on the challenges due to our parallel implementation.

4.3.1 Quadrature for integral equation

The domain boundary Γ is given by a collection of non-overlapping patches $\Gamma = \bigcup_i P_i(Q)$, where $P_i : Q \to \mathbb{R}^3$ is defined on $Q = [-1, 1]^2$. We use the Nyström discretization for Eq. (4.2.5). Since $D(\boldsymbol{x}, \boldsymbol{y})$ is singular, this requires a singular quadrature scheme for the integral on the right-hand side. We proceed in several steps, starting with the direct non-singular discretization, followed by an distinct discretization for the singular and near-singular case.

Non-singular integral quadrature. We discretize the integral in Eq. (4.2.5), for $\boldsymbol{x} \notin \Gamma$, by rewriting it as an integral over a set of patches and then apply a tensor-product qth order Clenshaw-Curtis rule to each patch:

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_i \int_{P_i} D(\boldsymbol{x}, \boldsymbol{y})\phi(\boldsymbol{y}) d\boldsymbol{y}_{P_i} \approx \sum_i \sum_{j=0}^{q^2} D(\boldsymbol{x}, \boldsymbol{y}_{ij}) w_{ij} \phi(\boldsymbol{y}_{ij}), \tag{4.3.1}$$

where $\boldsymbol{y}_{ij} = P_i(\boldsymbol{t}_j)$ and $\boldsymbol{t}_j \in [-1, 1]^2$ is the jth quadrature point and w_{ij} is the corresponding quadrature weight. We refer to the points $\boldsymbol{y}_{ij}$ as the *coarse discretization of* Γ and introduce a single global index $\boldsymbol{y}_\ell = \boldsymbol{y}_{ij}$ with $\ell = \ell(i, j) = (i - 1)q^2 + j$, $\ell = 1, \dots, N$, where N is the total number of quadrature nodes. We can then rewrite the right-hand side of (4.3.1) compactly as the vector dot product $W(\boldsymbol{x}) \cdot \phi$, where $\phi_\ell = \phi(\boldsymbol{y}_\ell)$ and $W_\ell(\boldsymbol{x}) = D(\boldsymbol{x}, \boldsymbol{y}_\ell) w_\ell$ are the quadrature weights in Eq. (4.3.1).

As $\boldsymbol{x} \to \Gamma$ for $\boldsymbol{x} \in \Omega$, the integrand becomes more singular and the accuracy of this quadrature rapidly decreases due to the singularity in the kernel D. This requires us to construct a singular integral discretization for $\boldsymbol{x} = \boldsymbol{y}_\ell$, $\ell = 1, \ldots, N$, and general points on Γ, which is discussed next. Note that the same method is used for evaluation of the velocity values at points close to the surface, once the equation is solved (*near-singular integration*).

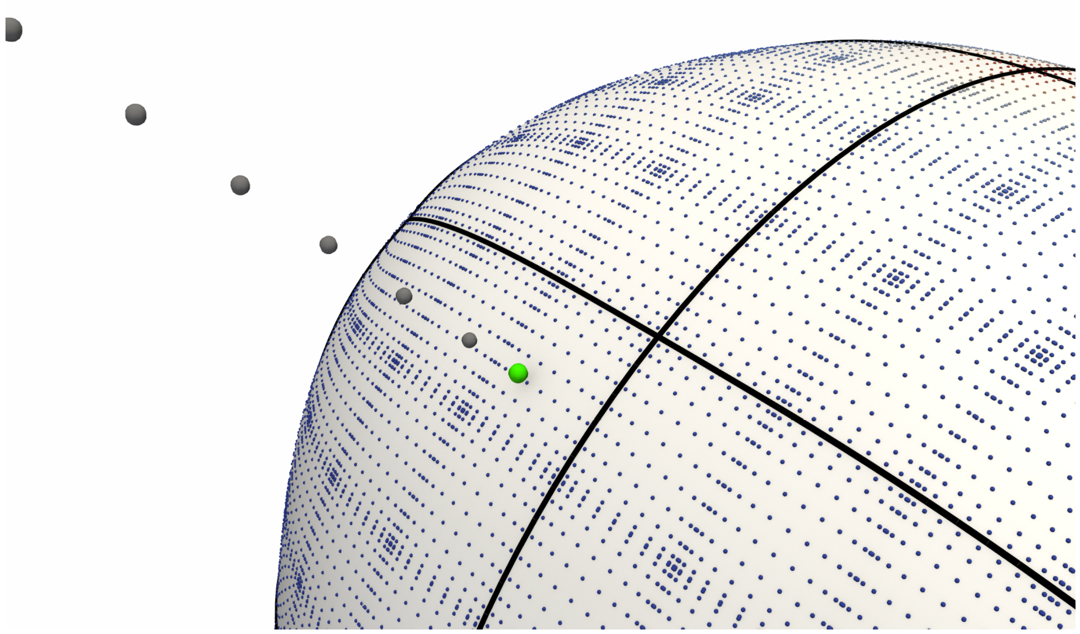


Figure 4.1: Boundary quadrature schematic. *Schematic of our unified singular/near-singular quadrature scheme. Shown is a domain boundary Γ split into patches (patch edges shown in black), an off-surface target point (green), check points (gray) and the fine discretization of Γ (small dots), which uses $k = 16$ tensor-product points per patch.*

Singular and near-singular integral quadrature. We take an approach similar to [45]. The idea is to evaluate the integral sufficiently far from the surface using the non-singular quadrature rule (4.3.1) on an upsampled mesh, and then to extrapolate the accurate values towards the surface. Concretely, to compute the singular integral at a point $\boldsymbol{x}$ near or on Γ, we use the following steps:

1. Upsample ϕ using qth order interpolation, i.e., $\phi^{\text{up}} = U\phi$, where ϕ^{up} is the

vector of Nk samples of the density and U is the interpolation operator. To be precise, we subdivide each patch P_i into k square subdomains P_{ik} and use Clenshaw-Curtis nodes in each subdomain. We subdivide uniformly, i.e., P_i is split into $k = 4^\eta$ patches for an integer η. This is the *fine discretization of* Γ. We use W^{up} to denote the weights for Eq. (4.3.1) the fine discretization quadrature points.

2. Find the closest point $\boldsymbol{y} = P(u^*, v^*)$ to $\boldsymbol{x}$ on Γ for some patch P on Γ with $u^*, v^* \in [-1, 1]$ ($\boldsymbol{y} = \boldsymbol{x}$ if $\boldsymbol{x} \in \Gamma$).

3. Construct *check points* $c_q = c_q(\boldsymbol{x}) = -(R + ir)\boldsymbol{n}(u^*, v^*)$, $i = 0, \dots, p$, where $\boldsymbol{n}(u, v)$ is the outward normal vector to Γ at $P(u, v)$.

4. Evaluate the velocity at the check points:

$$\boldsymbol{u}(c_q(\boldsymbol{x})) \approx W^{\mathrm{up}}(c_q) \cdot \phi^{\mathrm{up}}, \quad i = 0, \dots, p. \tag{4.3.2}$$

5. Extrapolate the velocity from the check points to $\boldsymbol{x}$ with 1D polynomial extrapolation:

$$\boldsymbol{u}(\boldsymbol{x}) \approx \sum_q e_q \boldsymbol{u}(c_q(\boldsymbol{x})) = \left(\sum_q e_q W^{\mathrm{up}}(c_q) \right) U\phi \tag{4.3.3}$$

$$= W^{\mathrm{s}}(\boldsymbol{x}) \cdot \phi, \tag{4.3.4}$$

where e_q are the extrapolation weights.

The parameters R, p, r and η are chosen to balance the error in the accuracy of $W^{\mathrm{up}}(c_q) \cdot \phi^{\mathrm{up}}$ and the extrapolation to $\boldsymbol{x}$. A schematic of this quadrature procedure is shown in Fig. 4.1.

Discretizing the integral equation. With the singular integration method described above, we take $\boldsymbol{x} = \boldsymbol{y}_\ell$, $\ell = 1 \dots N$, and obtain the following discretization of Eq. (4.2.5):

$$\left(\frac{1}{2}I + A\right)\phi = \boldsymbol{g}, \quad A_{\ell m} = W^{\mathrm{s}}_m(\boldsymbol{y}_\ell) + N_{ij}, \tag{4.3.5}$$

where $\boldsymbol{g}$ is the boundary condition evaluated at $\boldsymbol{y}_\ell$, $W^s_m(\boldsymbol{x})$ is the mth component of $W^s(\boldsymbol{x})$ and N_{ij} is the appropriate element of the rank-completing operator in Eq. (4.2.5).

The dense operator A is never assembled explicitly. We use GMRES to solve Eq. (4.3.5), which only requires application of A to vectors ϕ. This matrix-vector product is computed using the steps summarized above.

Extrapolation and upsampling are local computations that are parallelized trivially if all degrees of freedom for each patch are on a single processor. The main challenges in parallelization of the above singular evaluation are 1) initially distributing the patches among processors, 2) computing the closest point on Γ and 3) evaluating the velocity at the check points. The parallelization of these computations is detailed in the remainder of this section.

4.3.2 Distributing geometry and evaluation parallelization

We load pieces of the blood vessel geometry, which is provided as a quad mesh, separately on different processors. Each face of the quad mesh has a corresponding polynomial P_i defining the ith patch.

The k levels of patch subdivision induce a uniform quadtree structure within each quad. We use the `p4est` library [13] to manage this surface mesh hierarchy, keep track of neighbor information, distribute patch data and to refine and coarsen the discretization in parallel. The parallel quadtree algorithms provided by `p4est` are used to distribute the geometry without replicating the complete surface and polynomial patches across all processors. `p4est` also determines parent-child patch relationships between the coarse and fine discretizations and the coordinates of the child patches to which we interpolate.

Once the geometry is distributed, constructing check points, upsampling and extrapolation are performed locally.

4.3.3 Parallel closest point search

To evaluate the solution at a point $\boldsymbol{x}$, we must find the closest point $\boldsymbol{y}$ on the boundary to $\boldsymbol{x}$. The distance $\|\boldsymbol{x} - \boldsymbol{y}\|_2$ determines whether or not near-singular integration is required to compute the velocity at $\boldsymbol{x}$. If it is, $\boldsymbol{y}$ is used to construct check points.

In the context of this chapter, the point $\boldsymbol{x}$ is on the surface of an RBC, which may be on a different processor than the patch containing $\boldsymbol{y}$. This necessitates a parallel algorithm to search for $\boldsymbol{y}$. For that purpose, we extend the spatial sorting algorithm from [54, Algorithm 1] to support our fixed patch-based boundary and detect near pairs of target points and patches.

a. *Construct a bounding box $B_{P,\epsilon}$ for the near-zone of each patch.* We choose a distance d_ϵ so that for all points $\boldsymbol{z}$ further away than d_ϵ from P, the quadrature error of integration over P is bounded by ϵ. The set of points closer to P than d_ϵ is the *near-zone of* P. We inflate the bounding box B_P of P by d_ϵ along the diagonal to obtain $B_{P,\epsilon}$ to contain all such points.

b. *Sample $B_{P,\epsilon}$ and compute a spatial hash of the samples and $\boldsymbol{x}$.* Let H be the average diagonal length of all $B_{P,\epsilon}$. We sample the volume contained in $B_{P,\epsilon}$ with equispaced samples of spacing $h_P < H$. Using a spatial hash function, (such as Morton ordering, for a spatial grid with spacing H), we assign hash values to bounding box samples and $\boldsymbol{x}$ to be used as a sorting key. This results in a collection of hash values that contain the near-zone of Γ.

c. *Sort all samples by the sorting key.* Use the parallel sort of [98] on the sorting key of bounding box samples and that of $\boldsymbol{x}$. This collects all points with identical sorting key (i.e., close positions) and places them on the same processor. If the hash of $\boldsymbol{x}$ matches the hash of a bounding box sample, then $\boldsymbol{x}$ could require near-singular integration, which we check explicitly. Otherwise, we can assume $\boldsymbol{x}$ is sufficiently far from P and does not require singular integration.

d. *Compute distances $\|\boldsymbol{x} - P_i\|$.* For each patch P_i with a bounding box key of $\boldsymbol{x}$, we locally solve the minimization problem $\min_{(u,v)\in[-1,1]^2} \|\boldsymbol{x} - P_i(u,v)\|$ via Newton's method with a backtracking line search. This is a local computation since $\boldsymbol{x}$ and P_i were communicated during the Morton ID sort.

e. *Choose the closest patch P_i.* We perform a global reduce on the distances $\|\boldsymbol{x} - P_i\|$ to determine the closest P_i to $\boldsymbol{x}$ and communicate back all the relevant information required for singular evaluation back to $\boldsymbol{x}$'s processor.

4.3.4 Far evaluation and other quadrature algorithms

Far evaluation To compute the fluid velocity away from Γ, where Eq. (4.2.5) is non-singular, i.e., at the check points, the integral can be directly evaluated using Eq. (4.3.1). Observing that Eq. (4.3.1) has the form of an N-body summation, we use the *fast-multipole method* [31] to evaluate it for all target points at once. We use the parallel, kernel-independent implementation, PVFMM [56, 57], which has been demonstrated to scale to hundreds of thousands of cores. PVFMM handles all of the parallel communication required to aggregate and distribute the contribution of non-local patches in $O(N)$ time.

Other quadrature methods Various other parallel algorithms from are leveraged to perform boundary integration for the complex vessel geometry. To compute $\boldsymbol{u}^{\gamma_i}(\boldsymbol{X})$ for $\boldsymbol{X} \in \gamma_i$, the schemes presented in [105] are used to achieve spectral convergence for single-layer potentials by performing a spherical harmonic rotation and apply the quadrature rule of [30]. We use the improved algorithm in [58] to precompute the singular integration operator and improve overall complexity substantially. To compute $\boldsymbol{u}^{\gamma_i}(\boldsymbol{X})$ for $\boldsymbol{X}$ close to, but not on γ_i, we follow the approaches of [58, 95], which use a variation of the near-singular evaluation scheme of [118]. Rather than extrapolating the velocity from nearby check points as in Section 4.3, we use [105] to compute the velocity on surface, upsampled quadrature on γ_i to compute the velocity at check points and interpolate the velocity between them to the desired location.

4.4 Parallel collision handling

We preventing collisions of RBCs with other RBCs and with the vessel surface Γ by solving the NCP problem given in Eqs. (4.2.12) and (4.2.13). This is a nonsmooth and non-local problem, whose assembly and efficient solution is particularly challenging in parallel, especially in the context of complex geometries. In this section, we summarize our constraint-based approach and algorithm.

We have integrated piecewise polynomial patches into the framework of [54] for parallel collision handling, to which we refer the reader for a more detailed discussion. The key step to algorithmically unify RBCs and patches is to *form a linear triangle mesh approximation* of both objects. We now want to enforce that these meshes are collision-free subject to the physics constraints in Eq. (4.2.12).

We linearize the NCP and solve a sequence of LCP problems whose solutions converge to the NCP solution. At a high-level, the collision algorithm proceeds as follows:

1. Find triangle-vertex pairs of distinct meshes that are candidates for collision.
2. Compute $V(t^+) = V(t^{+,0})$. If any triangle-vertex pairs on distinct meshes collide, the corresponding component of $V(t)$ will be negative.
3. While $V_i(t^{+,k}) < 0$ for any i:
   (a) Suppose m components of $V(t)$ are negative
   (b) Solve the following linearized version of Eqs. (4.2.12) and (4.2.13)

$$\boldsymbol{X}_i^{+,k} = \boldsymbol{X}_i + \Delta t(\boldsymbol{b}_i + S_i(\mathbf{f}_i(\boldsymbol{X}_i^{+,k}, \lambda^{+,k})), \tag{4.4.1}$$

$$\lambda(t^+) \geq 0, \quad L(t^{+,k}) \geq 0, \quad \lambda(t^{+,k}) \cdot L(t^{+,k}) = 0, \tag{4.4.2}$$

$$\text{where} \quad L(t) = V(t) + \nabla_u V^T \Delta \boldsymbol{X}_i(t) \tag{4.4.3}$$

   for the kth iteration of the loop and $\boldsymbol{X}_i^{+,k} = \boldsymbol{X}_i + \Delta \boldsymbol{X}_i(t^{+,k})$.

   (c) Find new candidate triangle-vertex pairs and compute $V(t^{+,k})$.

Here, $t^{+,k}$ is the intermediate time step at which a new candidate position $\boldsymbol{X}_i^{+,k}$ occurs. Upon convergence of this algorithm, we are guaranteed that our system is collision-free.

To solve the LCP problem in Item 3b, we follow the approach detailed in [53, Section 3.2.2, Section 3.3]. We reformulate the problem first in standard LCP form with system matrix $\boldsymbol{B}$, then solve an equivalent root finding problem by applying a minimum-map Newton's method. This can be restructured to use GMRES, so we only need to repeatedly apply $\boldsymbol{B}$ to vectors to solve each LCP problem. Each entry $\boldsymbol{B}_{ij}$ is the change in the jth contact volume induced by the kth contact force, which is explicitly defined in [54, Algorithm 3]. This means that $\boldsymbol{B}$ is of size $m \times m$, where m is the number of collisions, but is extremely sparse. We need not store the entire matrix explicitly; we only compute the non-zero entries and store them in a distributed hash-map. Computing these matrix elements requires an accumulation of all coupled collision contributions to the velocity, which requires just a sparse `MPI_All_to_Allv` to send each local contribution to the process containing $V_i(t^{+,k})$.

An important step to ensure good scaling of our collision handling algorithm is to minimize the number of triangle-vertex pairs that are found in Item 1. One could explicitly compute an all-to-all collision detection on all meshes in the system, but this requires $O(N^2)$ work and global communication. We perform a high-level filtering first to find local *candidate collision mesh pairs*, then only communicate and compute the required $O(m)$ information. Since spatially-near mesh pairs may be on different processors, we need a parallel algorithm to compute these collision candidates.

To address this, we reuse Items a to c from Section 4.3.3 and adapt it to this problem. For each mesh in the system, we form the *space-time bounding box* of the mesh: the smallest axis-aligned bounding box containing the mesh at positions $\boldsymbol{X}_i$ and $\boldsymbol{X}_i^+$, as shown in Fig. 4.2. For patches P_i, note that $P_i^+ = P_i$. This means

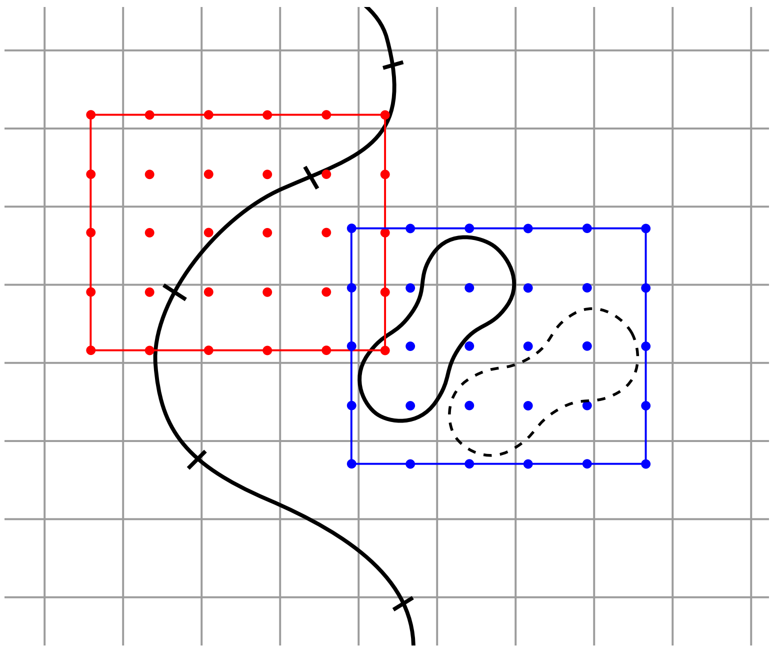


Figure 4.2: GRID FOR CANDIDATE COLLISION PAIR. *A 2D depiction of the parallel candidate collision pair algorithm. Shown is the implicit spatial grid (gray), a piece of the blood vessel* Γ *(open black curve), an RBC* γ_i *at the current time step (closed black curve) and at the next time step (dotted closed back curve). Also shown is the space-time bounding box and bounding box samples of a single patch (red square and red dots) and an RBC (blue square and blue dots).*

one can reuse the bounding box of P_i constructed in Section 4.3.3 for this purpose and simply set d_ϵ to zero. After forming all space-time bounding boxes for the meshes of all patches and RBCs, we apply steps Items b and c directly to these boxes. Item c will communicate meshes with the same spatial sorting key to the same processor; these meshes are collision candidate pairs. Once the computation is local and candidate collision pairs are identified, we can proceed with the NCP solution algorithm described above.

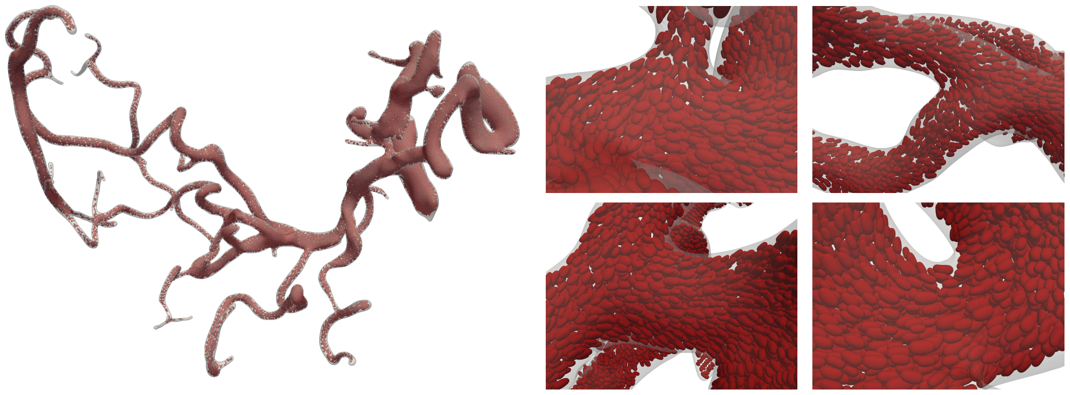


Figure 4.3: STRONG SCALABILITY SIMULATION DOMAIN. *Simulation results for 40,960 RBCs in a complex vessel geometry. For our strong scaling experiments, we use the vessel geometry shown on the left, with inflow-outflow boundary conditions at various regions of the vessel geometry. To setup the problem, we fill the vessel with nearly-touching RBCs of different sizes. The figure above shows a setup with overall 40,960 RBCs at a volume fraction of 19%, and 40,960 polynomial patches. The full simulation video is available at https://vimeo.com/329509229.*

4.5 Results

In this section, we present scalability results for our blood flow simulation framework on various test geometries. We also highlight simulations with various volume fractions, including one with volume fraction comparable to realistic human blood flows.

4.5.1 Implementation and example setup

Architecture and software libraries. We use the Stampede2 system at the Texas Advanced Computing Center (TACC) to study the scalability of our algorithms and implementation. Stampede2 has two types of compute nodes, the Knights Landing (KNL) compute nodes and the Skylake (SKX) compute nodes. The SKX cluster has 1,736 dual-socket compute nodes, each with two 24-core 2.1GHz CPUs and 192GB of memory. The KNL cluster has 4,200 compute nodes, with a

68-core Intel Xeon Phi 7250 1.4Ghz CPUs and 96GB of memory plus 16GB of high-speed MCDRAM. We run our simulations in a hybrid distributed-shared memory fashion: we run one MPI process per node, with one OpenMP thread per hardware core. Our largest simulations use 256 SKX and 512 KNL nodes.

We leverage several high-performance libraries in our implementation. We use PETSc's [5] parallel matrix and vector operations, and its parallel GMRES solver. Management and distribution of patches describing the blood vessel geometry uses the `p4est` library [13], and we use PVFMM [56] for parallel FMM evaluation. We also heavily leverage `Intel MKL` for fast dense linear algebra routines at the core of our algorithms and `paraview` for our visualizations.

Discretization and example setup For all test cases we present, we discretize each RBC with 544 quadrature points and 2,112 points for collision detection. The blood vessel geometry is represented with 8th order tensor-product polynomial patches with 121 quadrature points per patch and 484 equispaced points for collision detection.

Since our scaling tests are performed on complex, realistic blood vessel geometries, we must algorithmically generate our initial simulation configuration. We prescribe portions of the blood vessel as inflow and outflow regions and appropriately prescribe positive and negative parabolic flows (inlet and outlet flow) as boundary conditions, such that the total fluid flux is zero. To populate the blood vessel with RBCs, we uniformly sample the volume of the bounding box of the vessel with a spacing h to find point locations inside the domain at which we place RBCs in a random orientation. We then slowly increase the size of each RBC until it collides with the vessel boundary or another RBC; this determines a single RBC's size. We continue this process until all RBCs stop expanding; this means that we are running a simulation of RBCs of various sizes. We refer to this process as *filling the blood vessel with* RBCs. This is a precomputation for our simulation, so we do not include this step in the timings we report for weak and strong scaling.

Additionally, RBCs in such a confined flow will collide with the blood vessel wall if special care is not taken near the outflow part of the boundary. We define regions near the inlet and outlet flows where we can safely add and remove RBCs. When an RBC γ_i is within the outlet region, we subtract off the velocity due to γ_i from the entire system and move γ_i into an inlet region such that the arising RBC configuration is collision-free.

Limiting GMRES iterations We have observed that the GMRES solver typically requires 30 iterations or less for convergence for almost all time steps, but the number of needed iterations may vary more in the first steps. To simulate the amount of work in a typical simulation time step, we cap the number of GMRES iterations at 30 and report weak and strong scaling for these iterations. A more detailed analysis of this behavior is needed.

4.5.2 Parallel scalability

Strong scalability To study the strong scalability of our algorithms, we use the blood vessel geometry and RBC configuration in Fig. 4.3-left. This simulation contains 40,960 RBCs and the blood vessel is represented with 40,960 patches. With four degrees of freedom per RBC quadrature point and three per vessel quadrature point, this amounts to 89,128,960 and 14,868,480 degrees for the RBCs and blood vessel, respecitvely (103,997,440 in total). As can be seen from Fig. 4.5, we achieve

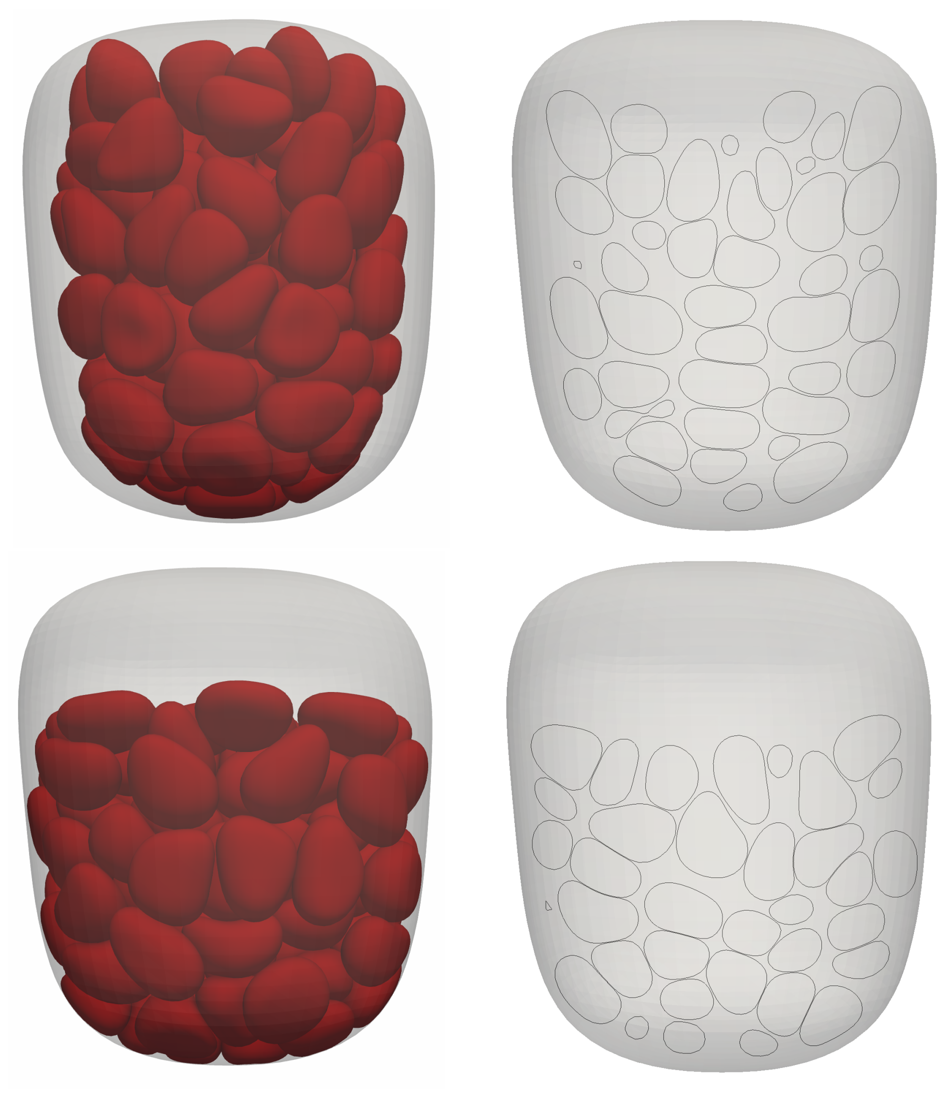
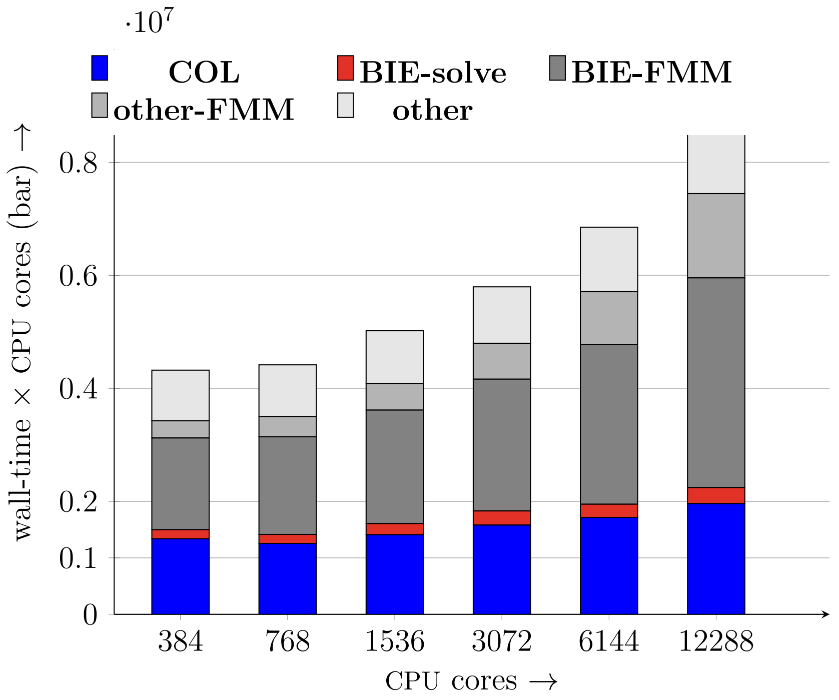


Figure 4.4: HIGH VOLUME FRACTION SIMULATION. *Shown is a high-volume fraction sedimentation due to gravitational force. The initial configuration (top figures) has a volume fraction of* 47%. *As the cells sediment to the lower part of the domain (bottom figures), the effective volume fraction of the final state is around* 55%. *Shown on the right side are slices through the center of the domain together with the RBC boundaries in the initial and final configuration. The full simulation video is available at https://vimeo.com/329509435.*

a 15.7-fold speed-up in total wall-time scaling from 384 to 12288 cores, corresponding to 49% parallel efficiency. This level of parallel efficiency is partially due to the calls to the fmm library PVFMM. The strong scalability of PVFMM we observe



cores	384	768	1536	3072	6144	12288
total time (sec)	11257	5751	3268	1887	1116	718
efficiency	1.00	0.98	0.86	0.75	0.63	0.49
COL+BIE-solve (sec)	3901	1843	1046	596	317	183
efficiency	1.00	1.05	0.93	0.82	0.77	0.66

Figure 4.5: STRONG SCALABILITY RESULT. *Strong scalability of a simulation with* 40960 *RBCs on Stampede's SKX partition for the vessel network geometry shown in Fig. 4.3. The vessel is discretized with* 40960 *polynomial patches. Shown in the bar graph is a breakdown of the compute resources (wall-time × CPU cores) required by the individual components for a simulation with 10 time steps on 384 to 12288 cores. The compute resources used by the main algorithms presented in this chapter are* ***COL*** *(collision handling),* ***BIE-solve*** *(computation of* $\mathbf{u}^\Gamma$*, not including FMM calls). Shown in different gray scales are the compute resources required by FMM (**BIE-FMM** and **FMM-other**) and other operations (**Other**). Shown in the table are the compute time and the parallel efficiency for the overall computation and for the sum of* ***COL*** *and* ***BIE-solve****. For the collision avoidance and the boundary solve we observe a parallel efficiency of* 66 % *for a 32-fold increase from 384 to 12288 CPU cores.*

is largely consistent with the results reported in [57]. Neglecting the time for calls to FMM, i.e., only counting the time for the boundary solver to compute $\boldsymbol{u}^\Gamma$ and

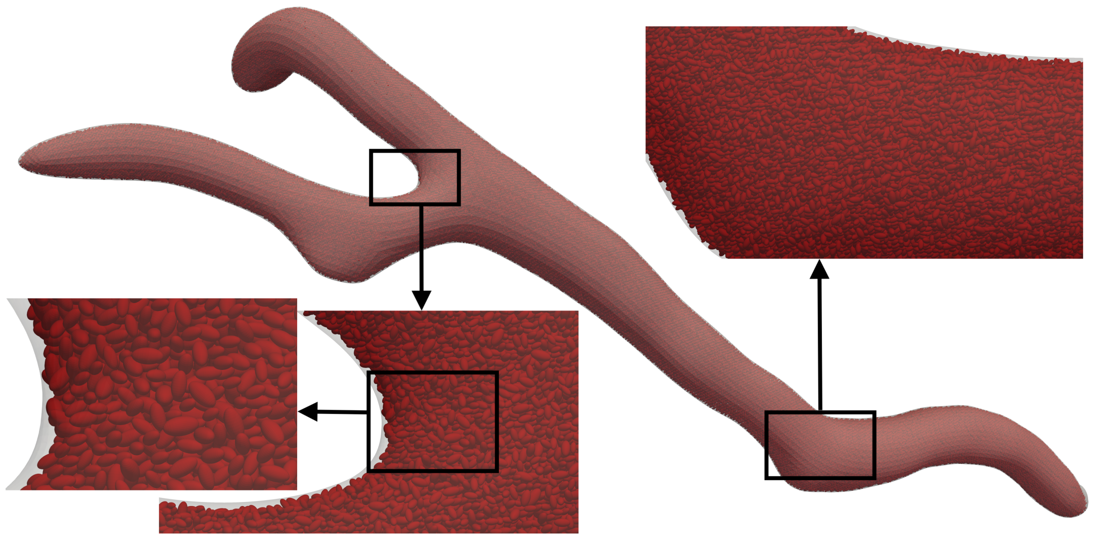
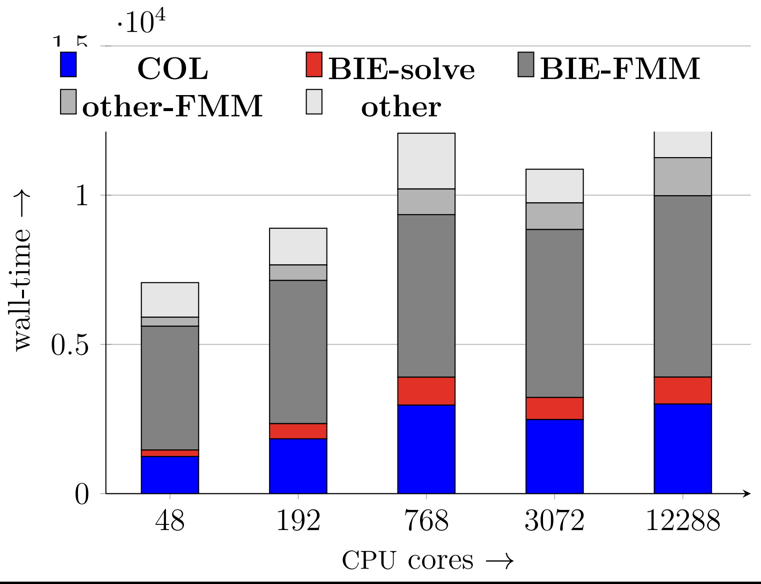


Figure 4.6: WEAK SCALABILITY SIMULATION DOMAIN. *For our weak scaling experiments, we use the the vessel geometry shown above with inflow boundary conditions on the right side and outflow boundary condition on the two left sides. To setup the problem, we fill the vessel with nearly-touching RBCs of different sizes to obtain a desired number, and refine the vessel geometry patches. The figure above shows a setup with overall 262,144 RBCs at a volume fraction of* 26%*.*

for collision prevention, we find 66% parallel efficiency when scaling strongly from 384 to 12288 cores.

Hence, the parallel collision handling and integral equation solver computations, excluding FMM, scale well as the number of cores is increased.

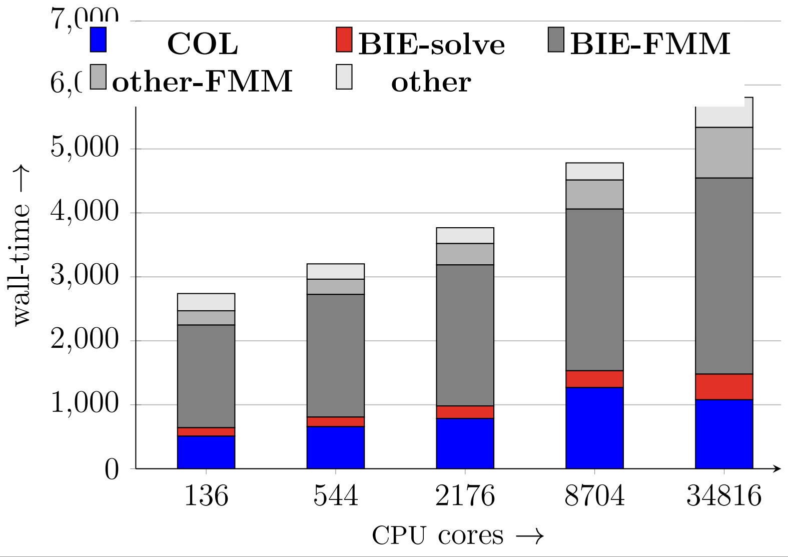
Weak Scalability Our weak scalability results are reported in Fig. 4.7 and Fig. 4.8. Both tests are performed on the blood vessel displayed in Fig. 4.6. We begin with an initial boundary composed of a fixed number M of polynomial patches and fill the domain with roughly $M/2$ RBCs (which requires spacing h). To scale up our simulation by a factor of four, we: (1) subdivide the M polynomial patches into $4M$ new but equivalent polynomial patches (via subdivision rules for Bezier curves); (2) refill the domain with RBCs using spacing $h/\sqrt[3]{4}$. This will place $2M$



cores	48	192	768	3072	12288
vol fraction	19%	20%	23%	26%	27%
#collision/ #RBCs	15%	13%	17%	15%	16%
total time (sec)	7070	8892	12076	10869	12446
efficiency	–	1.00	0.74	0.81	0.71
COL+BIE-solve (sec)	1461	2345	3901	3222	3904
efficiency	–	1.00	0.60	0.73	0.60

Figure 4.7: Weak scalability result. *Weak scalability on Stampede's SKX partition with node grain size of* 4096 *RBCs and* 8192 *polynomial patches per compute node (each node has 48 cores) for the vessel geometry shown in Fig. 4.6. Increasing the number of RBCs and boundary patches is realized by decreasing the size of the RBCs as discussed in Section 4.5.2. Shown in the bar graph is a breakdown of wall-time spent in individual components for a simulation with 10 time steps on 136 to 12288 cores (i.e., 4 to 256 nodes). The explanation of the labels used in the legend is detailed in Fig. 4.5. Additionally, we show the volume fraction of RBCs for each simulation, as well as the percentage of vesicles where the RBC-RBC or RBC-vessel collision prevention is active. We report the parallel scalability with respect to 192 cores, as the smallest simulation is in a single node and no MPI communication is necessary. The largest simulation has 1,048,576 RBCs and 2,097,152 polynomial patches and an overall number of 3,042,967,552 unknowns per time step.*

RBCs in the domain volume. We repeat this process each time we increase the number of cores by a factor of four in order to keep the number of patches and



cores	48	192	768	3072	12288
vol fraction	17%	19%	20%	23%	26%
#collision/ #RBCs	10%	15%	13%	17%	15%
total time (sec)	2739	3203	3768	4782	5806
efficiency	1.00	0.86	0.73	0.57	0.47
COL+BIE-solve (sec)	642	808	982	1532	1480
efficiency	1.00	0.79	0.65	0.42	0.43

Figure 4.8: WEAK SCALABILITY RESULT ON KNL NODES. *Same as Fig. 4.7 but on Stampede2's KNL partition with* 512 *RBCs and* 1024 *vessel boundary patches per node (each node has 68 cores). We find an overall parallel scalability of 47% for a 256-fold increase of the problem size.*

RBCs per core fixed.

The largest weak scaling test contains 1,048,576 RBCs and 2,097,152 polynomial patches on the blood vessel; we solve for 3,042,967,552 unknowns at each time step and are able to maintain a collision-free state between 4,194,304,000 triangular surface elements at each time step. Comparing the weak scalability results for SKX (Fig. 4.7) and KNL (Fig. 4.6), we observe similar qualitative behavior. Note that the smallest test on the SKX architecture only uses a single node, i.e., no MPI communication was needed. This explains the increased time for the collision prevention algorithms when going from 1 (48 cores) to 4 nodes (192 cores). Note also that the simulation on the KNL architecture used a significantly lower number of RBCs and geometry patches per node. Thus, this simulation has a larger ratio of communication to local work. This explains the less perfect scalability compared to the results obtained on the SKX architecture.

As with strong scaling, we see good parallel scaling of the non-FMM-related parts of the computation of $\boldsymbol{u}^{\Gamma}$ and the collision handling algorithm. The FMM-related calls dominate the overall runtime.

Discussion The parts of the algorithm introduced in this chapter scale as well as the FMM implementation we are using. However, our overall scaling is diminished by the multiple expensive FMM evaluations required for solving Eq. (4.2.5). This can be addressed by using a *local* singular quadrature scheme, i.e., compute a singular integral using the FMM on Eq. (4.3.1) directly, then compute a singular correction locally. This calculation has a three-fold impact on parallel scalability: (1) the FMM evaluation required is proportional to the size of the coarse discretization rather than the fine discretization ($O((p+1)N)$ vs. $O((k+p)N)$); (2) after the FMM evaluation, the local correction is embarrassingly parallel; (3) the linear operator Eq. (4.3.3) can be precomputed, making the entire calculation extremely fast with MKL linear algebra routines. These improvements together will allow our algorithm to scale well beyond the computational regime explored in this work.

4.5.3 High volume fraction

The RBC volume fraction, i.e., the ratio of volume occupied by RBCs compared to the overall blood volume is 35-45% in healthy women and 38-48% in healthy men.

As can be seen in the tables in Figs. 4.7 and 4.8, the volume fraction in our weak scaling simulations is below these values, which is due to the procedure used to fill the blood vessel with RBCs (see the discussion in Section 4.5.1). To demonstrate that we can simulate realistic high-volume fraction blood flows, Fig. 4.4 shows a test of 140 RBCs sedimenting under a gravitational force in a small capsule. The volume fraction for this example is 47%, calculated by dividing the amount of volume occupied by RBCs by the volume of the capsule. By the end of the simulation, we achieve an effective volume fraction of 55% (determined by bounding the RBCs by a cylinder) since the RBCs have become more tightly packed.

4.6 Conclusion

We have shown that our parallel platform for the simulation of red blood cell flows is capable of accurately resolved long-time simulation of red blood cell flows in complex vessel networks. We are able to achieve realistic cell volume fractions of over 47%, while avoiding collisions between cells or with the blood vessel walls. Incorporating blood vessels into red blood cell simulations requires solving a boundary integral equation, for which we use GMRES. Each GMRES iteration computes a matrix-vector product, which in turn involves singular quadrature and an FMM evaluation; the latter dominates the computation time. To avoid collisions, we solve a nonlinear complementarity problem in the implicit part of each time step. This requires repeated assembly of sparse matrices that, in principle, couple all cells globally. Nevertheless, solving this complementarity system yields close-to-optimal strong and weak scaling in our tests. Overall, the vast majority of compute time is spent in FMM evaluations, which implies that the scaling behavior of our simulation is dominated by the scalability of the FMM implementation. As discussed at the end of Section 4.5.2, in the future, we will employ a local singular quadrature scheme that will allow us to significantly reduce the time spent in FMM evaluations. This will not only speed up the overall simulation but also improve the weak and strong scalability of our simulation platform.

Bibliography

[1] B. Ahn. “Solution of nonsymmetric linear complementarity problems by iterative methods”. In: *Journal of Optimization Theory and Applications* 33.2 (1981), pp. 175–185.

[2] J. Allard, F. Faure, H. Courtecuisse, F. Falipou, C. Duriez, and P. G. Kry. “Volume contact constraints at arbitrary resolution”. In: *ACM SIGGRAPH 2010 papers on - SIGGRAPH ’10* C (2010), p. 1.

[3] B. K. Alpert. “Hybrid Gauss-trapezoidal quadrature rules”. In: *SIAM Journal on Scientific Computing* 20.5 (1999), pp. 1551–1584.

[4] U. M. Ascher, S. J. Ruuth, and B. T. Wetton. “Implicit-Explicit methods for time-dependent partial differential equations”. In: *SIAM Journal on Numerical Analysis* 32.3 (1995), pp. 797–823.

[5] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W Gropp, D. Kaushik, et al. *Petsc users manual revision 3.8.* Tech. rep. Argonne National Lab.(ANL), Argonne, IL (United States), 2017.

[6] P. Balogh and P. Bagchi. “A computational approach to modeling cellular-scale blood flow in complex geometry”. In: *Journal of Computational Physics* 334 (2017), pp. 280–307.

[7] P. Balogh and P. Bagchi. “Direct numerical simulation of cellular-scale blood flow in 3D microvascular networks”. In: *Biophysical journal* 113.12 (2017), pp. 2815–2826.

[8] D. Baraff and A. Witkin. “Large steps in cloth simulation”. In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’98* (1998), pp. 43–54.

[9] H. Basu, A. K. Dharmadhikari, J. A. Dharmadhikari, S. Sharma, and D. Mathur. “Tank treading of optically trapped red blood cells in shear flow”. In: *Biophysical Journal* 101.7 (2011), pp. 1604 –1612.

[10] A. R. Bausch and K. Kroy. “A bottom-up approach to cell mechanics”. In: *Nature Physics* 2.4 (2006), pp. 231–238.

[11] T. Biben, K. Kassner, and C. Misbah. “Phase-field approach to three-dimensional vesicle dynamics”. In: *Physical Review E* 72.4 (2005), p. 041921.

[12] O. P. Bruno and S. K. Lintner. “A high-order integral solver for scalar problems of diffraction by screens and apertures in three-dimensional space”. In: *Journal of Computational Physics* 252 (2013), pp. 250–274.

[13] C. Burstedde, L. C. Wilcox, and O. Ghattas. “`p4est`: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees”. In: *SIAM Journal on Scientific Computing* 33.3 (2011), pp. 1103–1133. DOI: `10.1137/100791634`.

[14] P. B. Canham. “The minimum energy of bending as a possible explanation of the biconcave shape of the human red blood cell”. In: *Journal of theoretical biology* 26.1 (1970), pp. 61–81.

[15] R. W. Cottle, J.-S. Pang, and R. E. Stone. *The linear complementarity problem*. Vol. 60. SIAM, 2009.

[16] V. Doyeux, S. Priem, L. Jibuti, A. Farutin, M. Ismail, and P. Peyla. “Effective viscosity of two-dimensional suspensions: Confinement effects”. In: *Physical Review Fluids* 1.4 (2016), p. 043301.

[17] P. Du, J. Zhao, W. Cao, and Y. Wang. “DCCD: Distributed N-Body Rigid Continuous Collision Detection for Large-Scale Virtual Environments”. In: *Arabian Journal for Science and Engineering* 42.8 (2017), pp. 3141–3147.

[18] Q. Du and J. Zhang. “Adaptive finite element method for a phase field bending elasticity model of vesicle membrane deformations”. In: *SIAM Journal on Scientific Computing* 30.3 (2008), pp. 1634–1657.

[19] C. Duriez, F. Dubois, A. Kheddar, and C. Andriot. “Realistic haptic rendering of interacting deformable objects in virtual environments.” In: *IEEE transactions on visualization and computer graphics* 12.1 (2006), pp. 36–47.

[20] C. Eck, O Steinbach, and W. Wendland. “A symmetric boundary element method for contact problems with friction”. In: *Mathematics and Computers in Simulation* 50.1 (1999), pp. 43–61.

[21] K. Erleben. “Numerical methods for linear complementarity problems in physics-based animation”. In: *ACM SIGGRAPH 2013 Courses* February (2013).

[22] A. Farutin, T. Biben, and C. Misbah. “3D Numerical simulations of vesicle and inextensible capsule dynamics”. In: (2013).

[23] F. Faure, S. Barbier, J. Allard, and F. Falipou. “Image-based collision detection and response between arbitrary volume objects”. In: *Proceedings of the 2008 ACM*. Vol. i. 2008, pp. 155–162.

[24] K. Fischer and P Wriggers. “Frictionless 2D contact formulations for finite deformations based on the mortar method”. In: *Computational Mechanics* 36.3 (2005), pp. 226–244.

[25] J. Freund. “Leukocyte margination in a model microvessel”. In: *Physics of Fluids (1994-present)* 19.2 (2007), p. 023301.

[26] J. B. Freund. “Numerical simulation of flowing blood cells”. In: *Annual review of fluid mechanics* 46 (2014), pp. 67–95.

[27] J. M. Frostad, J. Walter, and L. G. Leal. “A scaling relation for the capillary-pressure driven drainage of thin films”. In: *Physics of Fluids* 25.5 (2013), p. 052108.

[28] G. Ghigliotti, A. Rahimian, G. Biros, and C. Misbah. “Vesicle migration and spatial organization driven by flow line curvature”. In: *Physical Review Letters* 106.2 (Jan. 2011), p. 028101.

[29] J. Gounley, M. Vardhan, and A. Randles. “A Computational Framework to Assess the Influence of Changes in Vascular Geometry on Blood Flow”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. ACM. 2017, p. 2.

[30] I. G. Graham and I. H. Sloan. “Fully discrete spectral boundary integral methods for Helmholtz problems on smooth closed surfaces in $\mathbb{R}^3$”. In: *Numerische Mathematik* 92.2 (2002), pp. 289–323.

[31] L. Greengard and V. Rokhlin. “A fast algorithm for particle simulations”. In: *Journal of computational physics* 73.2 (1987), pp. 325–348.

[32] L. Grinberg, J. A. Insley, V. Morozov, M. E. Papka, G. E. Karniadakis, D. Fedosov, and K. Kumaran. “A new computational paradigm in multiscale simulations: Application to brain blood flow”. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM. 2011, p. 5.

[33] H Gun. “Boundary element analysis of 3-D elasto-plastic contact problems with friction”. In: *Computers & structures* 82.7 (2004), pp. 555–566.

[34] D. Harmon, E. Vouga, R. Tamstorf, and E. Grinspun. “Robust treatment of simultaneous collisions”. In: *ACM SIGGRAPH 2008 papers on - SIGGRAPH ’08* (2008), p. 1.

[35] D. Harmon, E. Vouga, B. Smith, R. Tamstorf, and E. Grinspun. “Asynchronous contact mechanics”. In: *ACM Transactions on Graphics* 28.3 (2009), p. 1.

[36] D. Harmon, D. Panozzo, O. Sorkine, and D. Zorin. “Interference-aware geometric modeling”. In: *ACM Transactions on Graphics* 30.6 (Dec. 2011), p. 1.

[37] W. Helfrich. “Elastic properties of lipid bilayers: theory and possible experiments”. In: *Zeitschrift für Naturforschung C* 28.11-12 (1973), pp. 693–703.

[38] K. Iglberger and U. Rüde. “A Parallel Rigid Body Dynamics Algorithm”. In: *Euro-Par 2009 Parallel Processing*. Ed. by H. Sips, D. Epema, and H.-X. Lin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 760–771.

[39] K. L. Johnson and K. L. Johnson. *Contact mechanics*. Cambridge university press, 1987.

[40] S. J. Karrila and S. Kim. “Integral equations of the second kind for Stokes flow: Direct solution for physical variables and removal of inherent accuracy limitations”. In: *Chemical Engineering Communications* 82.1 (1989), pp. 123–161.

[41] D. Kim, J.-P. Heo, J. Huh, J. Kim, and S.-e. Yoon. “HPCCD: Hybrid Parallel Continuous Collision Detection using CPUs and GPUs”. In: *Computer Graphics Forum* (2009).

[42] S. Kim and S. J. Karrila. *Microhydrodynamics: Principles and Selected Applications*. Courier Corporation, 2005.

[43] Y. Kim and M.-C. Lai. “Simulating the dynamics of inextensible vesicles by the penalty immersed boundary method”. In: *Journal of Computational Physics* 229.12 (June 2010), pp. 4840–4853.

[44] L. af Klinteberg and A.-K. Tornberg. “A fast integral equation method for solid particles in viscous flow using quadrature by expansion”. In: *Journal of Computational Physics* 326 (2016), pp. 420–445.

[45] A. Klöckner, A. Barnett, L. Greengard, and M. O’Neil. “Quadrature by expansion: A new method for the evaluation of layer potentials”. In: *Journal of Computational Physics* 252 (2013), pp. 332–349.

[46] R. H. Krause and B. I. Wohlmuth. “A Dirichlet–Neumann type algorithm for contact problems with friction”. In: *Computing and visualization in science* 5.3 (2002), pp. 139–148.

[47] A. Laadhari, P. Saramito, and C. Misbah. “Computing the dynamics of biomembranes by combining conservative level set and adaptive finite element methods”. In: *Journal of Computational Physics* 263 (2014), pp. 328–352.

[48] C. E. Lemke. “Bimatrix equilibrium points and mathematical programming”. In: *Management science* 11.7 (1965), pp. 681–689.

[49] E. Lerner, G. Düring, and M. Wyart. “A unified framework for non-Brownian suspension flows and soft amorphous solids”. In: *Proceedings of the National Academy of Sciences* 109.13 (2012), pp. 4798–4803. eprint: `http://www.pnas.org/content/109/13/4798.full.pdf`.

[50] F. Liu, T. Harada, Y. Lee, and Y. J. Kim. “Real-time Collision Culling of a Million Bodies on Graphics Processing Units”. In: *ACM SIGGRAPH Asia 2010 Papers*. SIGGRAPH ASIA ’10. Seoul, South Korea: ACM, 2010, 154:1–154:8.

[51] M. Loewenberg and E. J. Hinch. “Collision of two deformable drops in shear flow”. In: *Journal of Fluid Mechanics* 338 (May 1997), pp. 299–315.

[52] M. Loewenberg. “Numerical simulation of concentrated emulsion flows”. In: *Journal of Fluids Engineering* 120.4 (1998), p. 824.

[53] L. Lu, A. Rahimian, and D. Zorin. “Contact-aware simulations of particulate Stokesian suspensions”. In: *Journal of Computational Physics* 347C

(Nov. 2017), pp. 160–182. DOI: 10.1016/j.jcp.2017.06.039. arXiv: 1612.02057.

[54] L. Lu, A. Rahimian, and D. Zorin. “Parallel contact-aware simulations of deformable particles in 3D Stokes flow”. In: *arXiv preprint arXiv:1812.04719* (2018).

[55] L. Lu, M. Morse, A. Rahimian, G. Stadler, and D. Zorin. “Parallel contact-aware simulations of deformable particles in 3D Stokes flow”. In: *In preparation* (2019).

[56] D. Malhotra and G. Biros. “PVFMM: A Parallel Kernel Independent FMM for Particle and Volume Potentials”. In: *Communications in Computational Physics* 18 (2015), pp. 808–830.

[57] D. Malhotra and G. Biros. “Algorithm 967: A distributed-memory fast multipole method for volume potentials”. In: *ACM Transactions on Mathematical Software (TOMS)* 43.2 (2016), p. 17.

[58] D. Malhotra, A. Rahimian, D. Zorin, and G. Biros. “A parallel algorithm for long-timescale simulation of concentrated vesicle suspensions in three dimensions”. In: (2017).

[59] D. Malhotra, A. Rahimian, D. Zorin, and G. Biros. *A parallel algorithm for long-timescale simulation of concentrated vesicle suspensions in three dimensions.* 2018.

[60] O. Mangasarian. “Solution of symmetric linear complementarity problems by iterative methods”. In: *Journal of Optimization Theory and Applications* 22.4 (1977), pp. 465–485.

[61] H. Mazhar, T. Heyn, and D. Negrut. “A scalable parallel method for large collision detection problems”. In: 26 (June 2011), pp. 37–55.

[62] M. L. Minion et al. “Semi-implicit spectral deferred correction methods for ordinary differential equations”. In: *Communications in Mathematical Sciences* 1.3 (2003), pp. 471–500.

[63] C. Misbah. “Vacillating breathing and tumbling of vesicles under shear flow”. In: *Physical Review Letters* 96 (Jan. 2006), p. 028104.

[64] E. Nazockdast, A. Rahimian, D. Zorin, and M. Shelley. “Fast and high-order methods for simulating fiber suspensions applied to cellular mechanics”. preprint. 2015.

[65] M. Nemer, X. Chen, D. Papadopoulos, J. Bławzdziewicz, and M. Loewenberg. “Hindered and enhanced coalescence of drops in stokes flows”. In: *Physical Review Letters* 92.11 (Mar. 2004), p. 114501.

[66] J. Nocedal and S. Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[67] H. Noguchi and G. Gompper. “Vesicle dynamics in shear and capillary flows”. In: *Journal of Physics: Condensed Matter* 17.45 (Nov. 2005), S3439–S3444.

[68] R. Ojala and A.-K. Tornberg. “An accurate integral equation method for simulating multi-phase Stokes flow”. In: (Apr. 2014), pp. 1–22.

[69] M. a. Otaduy, R. Tamstorf, D. Steinemann, and M. Gross. “Implicit contact handling for deformable objects”. In: *Computer Graphics Forum* 28.2 (Apr. 2009), pp. 559–568.

[70] S. Pabst, A. Koch, and W. Strasser. “Fast and Scalable CPU/GPU Collision Detection for Rigid and Deformable Surfaces”. In: *Computer Graphics Forum* (2010).

[71] P. Perdikaris, L. Grinberg, and G. E. Karniadakis. “An effective fractal-tree closure model for simulating blood flow in large arterial networks”. In: *Annals of biomedical engineering* 43.6 (2015), pp. 1432–1442.

[72] P. Perdikaris, L. Grinberg, and G. E. Karniadakis. “Multiscale modeling and simulation of brain blood flow”. In: *Physics of Fluids* 28.2 (2016), p. 021304.

[73] M. Peyrounette, Y. Davit, M. Quintard, and S. Lorthois. “Multiscale modelling of blood flow in cerebral microcirculation: Details at capillary scale control accuracy at the level of the cortex”. In: *PloS one* 13.1 (2018), e0189474.

[74] H Power. “The completed double layer boundary integral equation method for two-dimensional Stokes flow”. In: *IMA Journal of Applied Mathematics* (1993).

[75] H. Power and G. Miranda. “Second kind integral equation formulation of Stokes’ flows past a particle of arbitrary shape”. In: *SIAM Journal on Applied Mathematics* 47.4 (1987), p. 689.

[76] C. Pozrikidis. “The axisymmetric deformation of a red blood cell in uniaxial straining Stokes flow”. In: *Journal of Fluid Mechanics* 216 (1990), pp. 231–254.

[77] C. Pozrikidis. *Boundary integral and singularity methods for linearized viscous flow.* Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 1992.

[78] C Pozrikidis. “Dynamic simulation of the flow of suspensions of two-dimensional particles with arbitrary shapes”. In: *Engineering Analysis with Boundary Elements* 25.1 (Jan. 2001), pp. 19–30.

[79] X. Provot. “Collision and self-collision handling in cloth model dedicated to design garments”. In: *Computer Animation and Simulation’97* (1997).

[80] M. A. Puso. “A 3D mortar method for solid mechanics”. In: *International Journal for Numerical Methods in Engineering* 59.3 (2004), pp. 315–336.

[81] M. A. Puso, T. Laursen, and J. Solberg. “A segment-to-segment mortar contact method for quadratic elements and large deformations”. In: *Computer Methods in Applied Mechanics and Engineering* 197.6 (2008), pp. 555–566.

[82] B. Quaife and G. Biros. “High-volume fraction simulations of two-dimensional vesicle suspensions”. In: *Journal of Computational Physics* 274 (2014), pp. 245–267.

[83] B. Quaife and G. Biros. “High-order adaptive time stepping for vesicle suspensions with viscosity contrast”. In: *Procedia IUTAM* 16 (2015), pp. 89–98.

[84] B. Quaife and G. Biros. “Adaptive time stepping for vesicle suspensions”. In: *Journal of Computational Physics* 306 (2016), pp. 478–499.

[85] M. Rachh and L. Greengard. “Integral equation methods for elastance and mobility problems in two dimensions”. In: *SIAM Journal on Numerical Analysis* 54.5 (2016), pp. 2889–2909.

[86] A. Rahimian, S. K. Veerapaneni, and G. Biros. “Dynamic simulation of locally inextensible vesicles suspended in an arbitrary two-dimensional do-

main, a boundary integral method". In: *Journal of Computational Physics* 229.18 (Sept. 2010), pp. 6466–6484.

[87] A. Rahimian, I. Lashuk, S. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, et al. "Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures". In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society. 2010, pp. 1–11.

[88] A. Rahimian, S. K. Veerapaneni, D. Zorin, and G. Biros. "Boundary integral method for the flow of vesicles with viscosity contrast in three dimensions". In: *Journal of Computational Physics* 298 (2015), pp. 766–786.

[89] J. Rallison and A. Acrivos. "A numerical study of the deformation and burst of a viscous drop in an extensional flow". In: *Journal of Fluid Mechanics* 89 (1978), pp. 191–200.

[90] A. Randles, E. W. Draeger, T. Oppelstrup, L. Krauss, and J. A. Gunnels. "Massively parallel models of the human circulatory system". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM. 2015, p. 1.

[91] D. Rossinelli, Y.-H. Tang, K. Lykov, D. Alexeev, M. Bernaschi, P. Hadjidoukas, M. Bisson, W. Joubert, C. Conti, G. Karniadakis, et al. "The in-silico lab-on-a-chip: petascale and high-throughput simulations of microfluidics at cell resolution". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM. 2015, p. 2.

[92] E. Sackmann. “Supported membranes: Scientific and practical applications”. In: *Science* 271 (1996), pp. 43–48.

[93] A. S. Sangani and G. Mo. “Inclusion of lubrication forces in dynamic simulations”. In: *Physics of Fluids* 6.5 (1994), pp. 1653–1662.

[94] J. S. Sohn, Y.-H. Tseng, S. Li, A. Voigt, and J. S. Lowengrub. “Dynamics of multicomponent vesicles in a viscous fluid”. In: *Journal of Computational Physics* 229.1 (2010), pp. 119–144.

[95] C. Sorgentone and A.-K. Tornberg. “A highly accurate boundary integral equation method for surfactant-laden drops in 3D”. In: *Journal of Computational Physics* 360 (2018), pp. 167–191.

[96] C. Sorgentone, A.-K. Tornberg, and P. M. Vlahovska. “A 3D boundary integral method for the electrohydrodynamics of surfactant-covered drops”. In: *Journal of Computational Physics* (2019).

[97] S Sukumaran and U. Seifert. “Influence of shear flow on vesicles near a wall: a numerical study”. In: *Physical Review E* 64.1 (2001), p. 011916.

[98] H. Sundar, D. Malhotra, and G. Biros. “HykSort: A New Variant of Hypercube Quicksort on Distributed Memory Architectures”. In: *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*. ICS ’13. Eugene, Oregon, USA: ACM, 2013, pp. 293–302.

[99] M. Tang, D. Manocha, and R. Tong. “Multi-core Collision Detection Between Deformable Models”. In: *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*. SPM ’09. San Francisco, California: ACM, 2009, pp. 355–360.

[100] M. Tang, D. Manocha, J. Lin, and R. Tong. “Collision-Streams: Fast GPU-based collision detection for deformable models”. In: *I3D ’11: Proceedings of the 2011 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. San Fransisco, CA, 2011, pp. 63–70.

[101] M. Tang, J. yi Zhao, R. feng Tong, and D. Manocha. “GPU accelerated convex hull computation”. In: *Computers & Graphics* 36.5 (2012). Shape Modeling International (SMI) Conference 2012, pp. 498 –506.

[102] M Tur, F. Fuenmayor, and P Wriggers. “A mortar-based frictional contact formulation for large deformations using Lagrange multipliers”. In: *Computer Methods in Applied Mechanics and Engineering* 198.37 (2009), pp. 2860–2873.

[103] S. K. Veerapaneni, D. Gueyffier, D. Zorin, and G. Biros. “A boundary integral method for simulating the dynamics of inextensible vesicles suspended in a viscous fluid in 2D”. In: *Journal of Computational Physics* 228.7 (2009), pp. 2334–2353.

[104] S. K. Veerapaneni, D. Gueyffier, G. Biros, and D. Zorin. “A numerical method for simulating the dynamics of 3D axisymmetric vesicles suspended in viscous flows”. In: *Journal of Computational Physics* 228.19 (Apr. 2009), pp. 7233–7249.

[105] S. K. Veerapaneni, A. Rahimian, G. Biros, and D. Zorin. “A fast algorithm for simulating vesicle flows in three dimensions”. In: *Journal of Computational Physics* 230.14 (2011), pp. 5610–5634.

[106] M. Wala and A. Klöckner. “A Fast Algorithm with Error Bounds for Quadrature by Expansion”. In: *arXiv preprint arXiv:1801.04070* (2018).

[107] M. Wala and A. Klöckner. “Optimization of Fast Algorithms for Global Quadrature by Expansion Using Target-Specific Expansions”. In: *arXiv preprint arXiv:1811.01110* (2018).

[108] M. S. Warren and J. K. Salmon. “A Parallel Hashed Oct-Tree N-body Algorithm”. In: *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*. Supercomputing ’93. Portland, Oregon, USA: ACM, 1993, pp. 12–21.

[109] P. Wriggers. “Finite element algorithms for contact problems”. In: *Archives of Computational Methods in Engineering* 2.4 (1995), pp. 1–49.

[110] P. Wriggers. *Computational contact mechanics*. Springer Science & Business Media, 2006.

[111] D. Xu, E. Kaliviotis, A. Munjiza, E. Avital, C. Ji, and J. Williams. “Large scale simulation of red blood cell aggregation in shear flows”. In: *Journal of Biomechanics* 46.11 (2013), pp. 1810–1817.

[112] W. Yan, H. Zhang, and M. J. Shelley. “Computing collision stress in assemblies of active spherocylinders: Applications of a fast and generic geometric method”. In: *The Journal of chemical physics* 150.6 (2019), p. 064109.

[113] A. Yazdani and P. Bagchi. “Three-dimensional numerical simulation of vesicle dynamics using a front-tracking method”. In: *Physical Review E* 85.5 (May 2012), p. 056308.

[114] T. Ye, L. Peng, and Y. Li. “Three-dimensional motion and deformation of a red blood cell in bifurcated microvessels”. In: *Journal of Applied Physics* 123.6 (2018), p. 064701.

[115] T. Ye, N. Phan-Thien, and C. T. Lim. “Particle-based simulations of red blood cellsA review”. In: *Journal of biomechanics* 49.11 (2016), pp. 2255–2266.

[116] T. Ye, N. Phan-Thien, C. T. Lim, L. Peng, and H. Shi. “Hybrid smoothed dissipative particle dynamics and immersed boundary method for simulation of red blood cells in flows”. In: *Physical Review E* 95.6 (2017), p. 063314.

[117] L. Ying, G. Biros, and D. Zorin. “A high-order 3D boundary integral equation solver for elliptic PDEs in smooth domains”. In: *Journal of Computational Physics* 219.1 (2006), pp. 247–275.

[118] L. Ying, G. Biros, and D. Zorin. “A high-order 3D boundary integral equation solver for elliptic PDEs in smooth domains”. In: *Journal of Computational Physics* 219.1 (2006), pp. 247–275.

[119] G. K. Youngren and A. Acrivos. “Stokes flow past a particle of arbitrary shape: a numerical method of solution”. In: *Journal of Fluid Mechanics* 69 (1975), pp. 377–403.

[120] H. Zhao and E. S. G. Shaqfeh. “The dynamics of a vesicle in simple shear flow”. In: *Journal of Fluid Mechanics* 674 (Mar. 2011), pp. 578–604.

[121] H. Zhao and E. S. G. Shaqfeh. “The dynamics of a non-dilute vesicle suspension in a simple shear flow”. In: *Journal of Fluid Mechanics* 725 (May 2013), pp. 709–731.

[122] H. Zhao, A. H. Isfahani, L. N. Olson, and J. B. Freund. “A spectral boundary integral method for flowing blood cells”. In: *Journal of Computational Physics* 229.10 (May 2010), pp. 3726–3744.

[123] H. Zhou and C. Pozrikidis. “The flow of ordered and random suspensions of two-dimensional drops in a channel”. In: *Journal of Fluid Mechanics* 255 (1993), pp. 103–127.

[124] A. Z. Zinchenko and R. H. Davis. “A boundary-integral study of a drop squeezing through interparticle constrictions”. In: *Journal of Fluid Mechanics* 564 (Sept. 2006), p. 227.

[125] A. Z. Zinchenko, M. M. A. Rother, and R. R. H. Davis. “A novel boundary-integral algorithm for viscous interaction of deformable drops”. In: *Physics of Fluids* 9.6 (1997), p. 1493.