CONTINUOUS ENERGY, SHARED MEMORY PARALLEL, 3-DIMENSIONAL WHOLE BLOOD SIMULATIONS

by

Andrew Kassen

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Mathematics

The University of Utah

August 2021

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Andrew Kassen**
has been approved by the following supervisory committee members:

Aaron L. Fogelson ,	Chair(s)	**6/28/2021** Date Approved
James P. Keener ,	Member	**6/22/2021** Date Approved
Robert M. Kirby ,	Member	**6/19/2021** Date Approved
Keith B. Neeves ,	Member	**6/17/2021** Date Approved
Varun Shankar ,	Member	**6/17/2021** Date Approved

by **Devar Khoshnevisan** , Chair/Dean of
the Department/College/School of **Mathematics**
and by **David B. Kieda** , Dean of The Graduate School.

ABSTRACT

Platelets play a critical role in repairing the vasculature as a major component of blood clots. Platelet simulations tend to focus on the motion of a single platelet in isolation interacting with a flat vessel wall or the tendency of platelets to marginate through interactions with red blood cells. This dissertation, in part, focuses on the interactions between platelets in a vessel lined by a bumpy wall.

We introduce a discretization of force densities on smoothly reconstructed surfaces based on analytic formulae. We derive methods for quadrature on homogeneous surfaces, such as the sphere. Using the geometry of the smooth surface reconstructions, we obtain weights for quadrature on smoothly perturbed surfaces. This results in a discrete set of surface forces. We present results for spherical harmonic- and radial basis function-based surface reconstructions.

We develop a fine-grained parallelization of the interaction operations used by the immersed boundary method. The algorithm avoids atomic operations and relies on the well-studied parallel key-value sort and segmented reduce primitives. The result is an algorithm that scales independently of the background grid. The algorithm exhibits nearly perfect scaling for increasing numbers of processors/threads and increasing problem size. The resulting method is appropriate for use on shared memory parallel machines, including general-purpose graphical processing units.

These two advances are used as components in immersed boundary simulations of whole blood, focusing on the interaction between red blood cells, platelets, and the endothelium. We consider the effect of endothelial geometry on platelet motion. We find red blood cells to be crucial for understanding the motion of platelets, to the point that wall geometry has a negligible effect. We observe behaviors for which the platelet remains near the endothelium for extended periods and red blood cell-mediated interactions between platelets and endothelium for which the platelet has reduced speed. We suggest these behaviors as mechanisms for vascular maintenance.

For Liz.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

First, I give my love to my wife and family. Through the good and the bad, they never wavered in their support. Without their continued encouragement, this dissertation would never have happened.

I thank my advisor, Aaron Fogelson, for the advice, empathy, and humor over the years. I am indebted to Varun Shankar and Aaron Barrett, whose interest and excitement for my work kept me motivated when I felt stagnant.

I wish to thank my office mates and good friends Greg Handy, Chris Miles, and Daniel Zavitz for eight years of good times despite the stress of grad school. I thank the pub quiz team, Sean McAfee and Kathy McNamara, for the weekly respite. I thank the members of the biofluids group for the weekly carrots. I hope that I was as helpful to you as you all were to me.

Finally, I thank the NSF for their support through grant DMS-1521748.

CHAPTER 1

INTRODUCTION

Blood is a complex mixture of cellular and fluid components, most notably composed of red blood cells (RBCs) and platelets suspended in plasma. RBCs are primarily a transport mechanism for oxygen throughout the body. Platelets, meanwhile, play a key role in the maintenance of the vasculature. Blood flows through vessels, which vary in diameter between a few centimeters in the aorta to several micrometers in the capillaries. The lumens of healthy vessels are lined by a single layer of endothelial cells, called the endothelium. Interactions between platelets and RBCs, and between platelets and the endothelium, are important for a platelet's function, but these interactions are typically studied in isolation. This chapter is devoted to the biological background that motivates this dissertation.

1.1 Red blood cells

At rest, RBCs are biconcave disk-shaped cells approximately 8 µm in diameter. The volume fraction occupied by RBCs, or *hematocrit*, ranges approximately from 36% to 45% in healthy humans. RBCs are anucleate, making them, in essence, sacs of hemoglobin, an oxygen-carrying metalloprotein. The hemoglobin within the RBC gives the cytoplasm a viscosity approximately five times that of the enclosing blood plasma. The biconcave disk shape gives RBCs a high surface area to volume ratio, thereby reducing the transport time of oxygen across the membrane.

In order to deliver oxygen throughout the body, RBCs must be extremely flexible, as some vessels are smaller than the cell itself. The cytoskeleton of an RBC is composed of spectrin arranged in a sparse hexagonal mesh, which allows for such large deformations. Due to the wide variety of shapes exhibited by RBCs, their mechanical properties were studied intently during the 1970s and 80s. Canham theorized that the biconcave disk shape minimizes bending energy [2]. Skalak *et al.* devised a purpose-built constitutive law to describe the tension of RBC membranes [15]. Under the assumption of a viscoelastic response, Evans & Hochmuth estimated the membrane viscosity [5]. Mohandas & Evans gave estimates of the shear, bulk, and bending moduli [13], which have guided RBC models ever since.

At low shear rates, dilute suspensions of otherwise highly deformable RBCs behave as a rigid particle and tumble in the flow direction. At higher shear rates RBCs take on an elongated, convex shape. They incline at an angle above horizontal, and the membrane rotates around its internal fluid while maintaining roughly the same shape. Numerous studies use these two behaviors as validation of their RBC models [6, 14, 19, 20]. At physiological hematocrits, red blood cells tend toward the center of the blood vessel.

1.2 Platelets

Platelets in their inactive state are ellipsoidal disks, approximately 3–4 µm by 1 µm in size. They are much more rigid than RBCs, a property afforded to the platelet by its small size. Platelets are also much less prevalent, with 10–20 RBCs per platelet. In whole blood, the size and relative rigidity of platelets contribute to the *margination* of platelets: as RBCs migrate to the center of the vessel, they exclude platelets vessel, resulting in a "cell-free" layer near the wall that is devoid of RBCs and inhabited primarily by platelets. The size of this region depends on the size of the vessel, the shear rate, and the hematocrit, but is typically a few microns in size.

Less is known about the mechanical properties of platelets. Models range from perfectly rigid ellipsoids [17] to systems of springs with [4, 16] or without [18] a preferred curvature. One study estimates the shear modulus and viscosity for platelets [8], but models tend to use a higher shear modulus than estimated and neglect viscous effects altogether.

Platelets travel along the vasculature and respond to vascular injury. This response involves structural changes and the release of chemical signals, a process known as *activation*. These signals activate other platelets and recruit them to the injury site, while also setting off a complex cascade of chemical reactions to ultimately form fibrin. Activated platelets that come into contact with one another form an aggregate, which, together with the fibrin, form a clot. The clot prevents the loss of blood, and the ability to transport oxygen. However, the chemicals advect with the fluid, washing downstream in an instant. We have a paradox: how can the chemical signals activate upstream platelets while simultaneously being swept away [7]? At low shear rates, chemical diffusion may suffice. In pathological shear rates, platelets activate in response to increased shear rate [9]. This leaves only intermediate shear conditions. The prevailing theory relies on fast bonding to slow the platelet for activation and strong bonds to form [10]. Modeling the near-wall dynamics of platelets is therefore of the utmost interest.

1.3 Endothelium

Endothelial cells line the entirety of the vessel. They are tightly packed and bound to the subendothelium, as exposure of the subendothelium to platelets triggers their activation. Endothelial cells contain a rigid nucleus. The nucleus gives the endothelial cell a shape that protrudes into the lumen of the blood vessel, giving it a bumpy texture. This protrusion is approximately 1 µm in height, roughly the minor axis length of a platelet. Since platelets tend to occupy the space near the endothelium, this may have a marked effect on the dynamics of the platelets.

Few details about the mechanical properties of the endothelium are known. Models for which the interior of the endothelial cell is an elastic solid give a wide range of possible parameters [1]. The alignment of endothelial cells is dependent upon the shear rate of the blood plasma. Experiments with cow [3, 12] and dog [11] aortas illustrate the dependence of endothelial cell shape on the shear stress. Endothelial cells exposed to a low shear arrange into a cobblestone-like pattern with a footprint aspect ratio of approximately 1:1. Under higher shear stress, the cells elongate in the flow direction, with some cells exhibiting much higher aspect ratios. Simulations of blood tend to ignore the geometry of the endothelium entirely, instead opting for a flat plane or smooth tube. We model the endothelium with a more physiologically relevant shape and aim to determine what effect, if any, the shape has on the motion of nearby platelets.

The remainder of this dissertation starts from a standard view of the immersed boundary method, which is a popular tool for fluid-structure interaction. Chapter 2 gives preliminary details of the immersed boundary method and geometry on surface reconstructions. The study presented in Chapter 3 is a collaboration with Ondrej Maxian and Wanda Strychalski and details the calculation of membrane forces from smooth membrane reconstructions. In Chapter 4, we introduce a novel parallelization scheme for the interaction operations of the IB method. We apply these methods to the problem of whole blood simulation in Chapter 5. Finally, we summarize the contributions of this dissertation and give future directions in Chapter 6.

References

[1] N. Caille, O. Thoumine, Y. Tardy, and J.-J. Meister, *Contribution of the nucleus to the mechanical properties of endothelial cells*, Journal of Biomechanics, 35 (2002), pp. 177–187.

[2] P. B. Canham, *The minimum energy of bending as a possible explanation of the biconcave shape of the human red blood cell*, Journal of Theoretical Biology, 26 (1970), pp. 61–81.

[3] C. F. Dewey Jr, S. R. Bussolari, M. A. Gimbrone Jr, and P. F. Davies, *The dynamic response of vascular endothelial cells to fluid shear stress*, Journal of Biomechanical Engineering, 103 (1981), pp. 177–185.

[4] L. C. Erickson and A. L. Fogelson, *Computational model of whole blood exhibiting lateral platelet motion induced by red blood cells*, International Journal for Numerical Methods in Biomedical Engineering, 26 (2010), pp. 471–487.

[5] E. A. Evans and R. M. Hochmuth, *Membrane viscoelasticity*, Biophysical Journal, 16 (1976), pp. 1–11.

[6] T. G. Fai, B. E. Griffith, Y. Mori, and C. S. Peskin, *Immersed boundary method for variable viscosity and variable density problems using fast constant-coefficient linear solvers I: numerical method and results*, SIAM Journal on Scientific Computing, 35 (2013), pp. B1132–B1161.

[7] A. L. Fogelson and K. B. Neeves, *Fluid mechanics of blood clot formation*, Annual Review of Fluid Mechanics, 47 (2015), pp. 377–403.

[8] J. H. Haga, A. J. Beaudoin, J. G. White, and J. Strony, *Quantification of the passive mechanical properties of the resting platelet*, Annals of Biomedical Engineering, 26 (1998), pp. 268–277.

[9] P. A. Holme, U. Ørvim, M. J. A. G. Hamers, N. O. Solum, F. R. Brosstad, R. M. Barstad, and K. S. Sakariassen, *Shear-induced platelet activation and platelet microparticle formation at blood flow conditions as in arteries with a severe stenosis*, Atherosclerosis, Thrombosis, and Vascular Biology, 17 (1997), pp. 646–653.

[10] S. P. Jackson, *The growing complexity of platelet aggregation*, Blood, 109 (2007), pp. 5087–5095.

[11] M. J. Levesque, D. Liepsch, S. Moravec, and R. M. Nerem, *Correlation of endothelial cell shape and wall shear stress in a stenosed dog aorta.*, Arteriosclerosis, Thrombosis, and Vascular Biology, 6 (1986), pp. 220–229.

[12] M. J. Levesque and R. M. Nerem, *The elongation and orientation of cultured endothelial cells in response to shear stress*, Journal of Biomechanical Engineering, 107 (1985), pp. 341–347.

[13] N. Mohandas and E. A. Evans, *Mechanical properties of the red cell membrane in relation to molecular structure and genetic defects*, Annual Review of Fluid Mechanics, 23 (1994), pp. 787–818.

[14] T. Omori, T. Ishikawa, D. Barthès-Biesel, A. V. Salsac, Y. Imai, and T. Yamaguchi, *Tension of red blood cell membrane in simple shear flow*, Physical Review E, 86 (2012), pp. 056321–056329.

[15] R. Skalak, A. Tozeren, R. P. Zarda, and S. Chien, *Strain energy function of red blood cell membranes*, Biophysical Journal, 13 (1973), pp. 245–264.

[16] T. Skorczewski, L. C. Erickson, and A. L. Fogelson, *Platelet motion near a vessel wall or thrombus surface in two-dimensional whole blood simulations*, Biophysical Journal, 104 (2013), pp. 1764–1772.

[17] W. Wang, T. G. Diacovo, J. Chen, J. B. Freund, and M. R. King, *Simulation of platelet, thrombus and erythrocyte hydrodynamic interactions in a 3D arteriole with in vivo comparison*, PLoS ONE, 8 (2013), p. e76949.

[18] Z. Wu, Z. Xu, O. V. Kim, and M. S. Alber, *Three-dimensional multi-scale model of deformable platelets adhesion to vessel wall in blood flow*, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 372 (2014), p. 20130380.

[19] Y.-Q. Xu, F.-B. Tian, and Y.-L. Deng, *An efficient red blood cell model in the frame of IB-LBM and its application*, International Journal of Biomathematics, 6 (2013), p. 1250061.

[20] A. Z. K. Yazdani, R. M. Kalluri, and P. Bagchi, *Tank-treading and tumbling frequencies of capsules and red blood cells*, Physical Review E, 83 (2011), p. 046305.

CHAPTER 2

NUMERICAL METHODS AND MODELS

Blood plasma is a mixture of primarily water, proteins, and other small molecules and electrolytes. It therefore behaves like water, *i.e.*, an incompressible Newtonian fluid, with density $\rho = 1\,\mathrm{g/cm^3}$ and viscosity $\mu = 0.1\,\mathrm{cP}$. The plasma velocity field evolves according to the incompressible Navier-Stokes equations for Newtonian fluids,

$$\rho(\boldsymbol{u}_t + \nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u})) = \mu \nabla^2 \boldsymbol{u} - \nabla p + \boldsymbol{f}, \tag{2.1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2.2}$$

where $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}, t)$ is the fluid velocity, $p = p(\boldsymbol{x}, t)$ is the pressure, and $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{x}, t)$ is an external force density, each at location $\boldsymbol{x} = (x, y)$ in two dimensions and $\boldsymbol{x} = (x, y, z)$ in three, and at time t. In two dimensions, $\boldsymbol{u}$ has components (u, v), and in three dimensions, (u, v, w). The advection in Equation (2.1) is written in conservative form, and $\otimes$ denotes the outer product. Though we do not consider the two-dimensional case in this dissertation, we describe them for completeness and for ease of exposition. The methods are also equally applicable to two-dimensional problems.

RBCs and platelets are other major constituents of blood, and together with the endothelium are modeled as elastic interfaces. Each of these cell types has its own mechanical properties. Let Γ denote one such cell membrane, which has parametrization $\boldsymbol{X}(\theta, \varphi, t)$ at material coordinates (θ, φ) and time t. We define the energy functional

$$\mathcal{E}[\boldsymbol{X}] = \int_\Gamma W \, \mathrm{d}\boldsymbol{X}, \tag{2.3}$$

where W is an energy density function or constitutive law specific to the cell type, and is a function of $\boldsymbol{X}$ or its derivatives. The force density on the cell membrane is found via

$$\boldsymbol{F} = -\delta\mathcal{E}, \tag{2.4}$$

where δ denotes the first variation.

This chapter covers the minor implementation details, such as parallelization provided through library functions, and mathematical preliminaries which will be useful in the

upcoming chapters. These include the method of solving the Navier-Stokes equations, the elastic models for the cell membranes, the method of constructing cell surfaces from point clouds, and an overview of the immersed boundary method.

2.1 Solution of the incompressible Navier-Stokes equations

This section describes the fluid solver used in Chapters 4 and 5. Its design is recursive so that it is equally applicable to problems in two or three dimensions. For ease of exposition, we illustrate the method in two dimensions. However, the method will ultimately be applied to three-dimensional blood flow.

Let Ω be a rectangular d-dimensional ($d = 2$ or 3) domain, filled with plasma. We discretize Ω into a regular grid of rectilinear cells with side length h. We refer to this grid as the background grid. To discretize Equations (2.1) and (2.2), we use a marker-and-cell (MAC) grid [35]: denoting the center of cell i by $\boldsymbol{x}_i$, scalar-valued function $s(\boldsymbol{x})$ is discretized at $\boldsymbol{x}_i$, and component $\boldsymbol{e}_a \cdot \boldsymbol{v}$ of vector-valued function $\boldsymbol{v}(\boldsymbol{x})$ at $\boldsymbol{x}_i - 1/2h\boldsymbol{e}_a$, for $a = 1, \ldots, d$, where $\boldsymbol{e}_a$ is a canonical basis vector for $\mathbb{R}^d$. Figure 2.1(a) and (b) show the discretization locations for $\boldsymbol{u}$ in two dimensions. For fixed $\boldsymbol{e}_a$, we define Ω_h^a as the set of points at which $\boldsymbol{e}_a \cdot \boldsymbol{v}$ is discretized. These points are staggered relative to the background grid by $h\boldsymbol{g}_a = 1/2h(\mathbf{1} - \boldsymbol{e}_a)$, where $\mathbf{1}$ is a vector of d ones. We refer to $\boldsymbol{g}_a$ as the *staggering* of Ω_h^a. Let n_ω^a be the number of points in Ω_h^a, which may differ from the number of vertices on the background grid or the size of the other such sets. Each grid point $\boldsymbol{x} \in \Omega_h^a$ decomposes into $\boldsymbol{x} = h(\boldsymbol{i} + \boldsymbol{g}_a)$, where the components of $\boldsymbol{i}$ are integers. While there is one such set per spatial dimension, we will often consider only one at a time, which we will refer to simply as Ω_h, its cardinality as n_ω, and its staggering as $\boldsymbol{g}$.

Define the centered difference operators

$$D_a\phi(\boldsymbol{x}) = \frac{\phi(\boldsymbol{x} + 1/2h\boldsymbol{e}_a) - \phi(\boldsymbol{x} - 1/2h\boldsymbol{e}_a)}{h}, \quad a = 1, \ldots, d,$$

for which, *e.g.*, D_1 approximates a derivative in the x direction. The discrete divergence, gradient, and Laplacian are built upon this operator, resulting in a 2-point stencil for each discrete first derivative and the standard $2d + 1$-point discrete Laplacian. By averaging u in the y direction and v in the x direction, we obtain collocated approximations to u and v at the center of a cell edge. Averaging, *e.g.*, u in the x direction or v in the y direction yields an approximation at the cell center. We use the averaged values to construct the values of $\boldsymbol{u} \otimes \boldsymbol{u}$. We define the centered average operators

$$A_a\phi(\boldsymbol{x}) = \frac{\phi(\boldsymbol{x} + 1/2h\boldsymbol{e}_a) + \phi(\boldsymbol{x} - 1/2h\boldsymbol{e}_a)}{2}, \quad a = 1, \ldots, d.$$

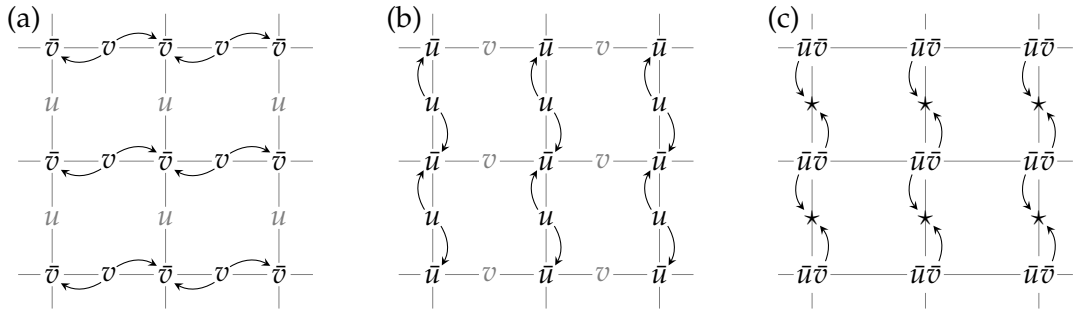


Figure 2.1: A cross-section illustrating the steps in computing $D_2[(A_2u)(A_1v)]$ from the first component in the discrete advection. The horizontal and vertical velocity component discretization locations are marked by u and v, respectively. Arrows emanate from points contributing to a stencil and point to the center of the stencil. (a) A_1 averages v in the x direction, yielding an approximation $\bar{v}$ at grid vertices (in 3D, centers of cell edges for which x and y are constant). (b) A_2 averages u in the y direction, yielding an approximation $\bar{u}$ at the same points as (a). The quantities A_1v and A_2u are collocated and can be directly multiplied to obtain an approximation of uv at locations marked $\bar{u}\bar{v}$. (c) D_2 approximately differentiates uv in the y direction, yielding the desired quantity at each point marked $\star$. The approximation of uv is also used to compute $D_1[(A_1v)(A_2u)]$ in the second component of the advection, wherein application of D_1 instead yields approximations collocated with locations marked v in (a).

We then discretize the components of the three-dimensional advection term by

$$\nabla_h \cdot (\boldsymbol{u} \otimes \boldsymbol{u}) := \begin{bmatrix} D_1[(A_1u)(A_1u)] + D_2[(A_1v)(A_2u)] + D_3[(A_1w)(A_3u)] \\ D_1[(A_2u)(A_1v)] + D_2[(A_2v)(A_2v)] + D_3[(A_2w)(A_3v)] \\ D_1[(A_3u)(A_1w)] + D_2[(A_3v)(A_2w)] + D_3[(A_3w)(A_3w)] \end{bmatrix}. \quad (2.5)$$

The symbol $\nabla_h \cdot$ denotes the discrete divergence operator. Figure 2.1 illustrates the steps in computing $D_1[(A_1v)(A_2u)]$, which appears in the first component in Equation (2.5). The two-dimensional case is identical, but with the last row and column (terms involving A_3 or D_3) deleted, and off-diagonal elements of $\boldsymbol{u} \otimes \boldsymbol{u}$ approximated at grid vertices. Analysis by Morinishi *et al.* show that this scheme, *Div.* $-$ *S*2 in their parlance, is conservative under the assumption that $\boldsymbol{u}$ is discretely divergence-free [38].

To advance the solution, we use an implicit-explicit Runge-Kutta (RK) method. We use a first-order method based on either the 1-stage backward-forward Euler scheme [22] or the 2-stage scheme described by Peskin [41]. For the backward-forward Euler scheme, discretizing Equation (2.1) to advance time from t to $t + \Delta t$ yields linear solves of Helmholtz type,

$$(I - \Delta t \rho^{-1} \mu \nabla_h^2)\boldsymbol{u}^* = \boldsymbol{u}^n + \Delta t \left[\rho^{-1}(\boldsymbol{f}^{n+1} - \nabla_h p^n) - \nabla_h \cdot (\boldsymbol{u}^n \otimes \boldsymbol{u}^n)\right] \quad \text{in } \Omega, \quad (2.6)$$

with boundary conditions

$$\boldsymbol{u}^* = \boldsymbol{u}_b^{n+1} + \Delta t \nabla_h q^n \quad \text{on } \partial\Omega, \quad (2.7)$$

where superscripts indicate the time step, $\boldsymbol{u}_b$ is velocity boundary data, and ∇_h^2 and ∇_h are the discrete Laplacian and gradient, respectively. The force density $\boldsymbol{f}^{n+1}$ is advanced explicitly, as will be discussed in Section 2.4. The intermediate velocity field $\boldsymbol{u}^*$ may not be divergence-free. To obtain a velocity field that is discretely divergence free, we use projection method II (PmII) of Brown *et al.* [24]. PmII updates the pressure,

$$p^{n+1} = p^n + (\rho I + \Delta t \mu \nabla_h^2) q^{n+1},$$

and generates the divergence-free velocity field

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^* - \Delta t \nabla_h q^{n+1} \tag{2.8}$$

using pseudo-pressure q^{n+1}, which satisfies

$$\begin{aligned} \Delta t \nabla_h^2 q^{n+1} &= \nabla_h \cdot \boldsymbol{u}^* \quad \text{in } \Omega \\ \boldsymbol{n} \cdot \nabla_h q^{n+1} &= 0 \qquad\quad \text{on } \partial\Omega. \end{aligned} \tag{2.9}$$

The velocity update (2.8) provides the boundary conditions (2.7) using a lagged value of the pseudo-pressure. The 2-stage RK method consists of a backward-forward Euler step followed by a Crank-Nicolson-midpoint step, which involves only minor modifications to Equation (2.6). In total, we perform 3 Helmholtz solves and 1 Poisson solve per RK stage.

We employ preconditioned conjugate gradients (PCG) to perform the solves. We precondition the Helmholtz solves with Chebyshev iteration. We precondition the Poisson solve with multigrid (MG) whose error smoothing procedure and direct solver are also based on Chebyshev iteration. Chebyshev iteration is a generalization of weighted Jacobi, and is effective in reducing PCG iterations compared to Gauss-Seidel iteration [21]. Given a range $0 \leq \lambda_1 \leq \lambda_2$, Chebyshev iteration uses a matrix polynomial derived from a Chebyshev polynomial to damp eigenvalue modes between λ_1 and λ_2. Because the Helmholtz and Poisson systems involve symmetric matrices, we use Geršgorin's theorem to find bounds on the matrix spectrum. These bounds are used directly in Chebyshev iteration preconditioning, while λ_1 is chosen to conservatively separate low and high-frequency modes for MG error smoothing. This λ_1 also recovers the optimal weight for Jacobi iteration (Chebyshev iteration derived from $T_2(x) = 2x^2 - 1$) for equilateral domains. Chebyshev iteration requires only the ability to perform sparse matrix polynomial-vector multiplication. Chebyshev iteration (MG) PCG is therefore parallelized by iterated application of a parallel sparse matrix-vector multiplication routine within Horner's method. We use `cuSPARSE` from NVIDIA® to provide sparse matrix-vector multiplication [39]. Convergence to tolerance $\tau = 10^{-11}$ is typically achieved in $\mathcal{O}(10)$ iterations for time steps ranging from $\Delta t = 10^{-8}$ to

10^{-6} sec for Helmholtz solves, and $\mathcal{O}(1)$ for the MG on grids presented in this dissertation. However, MG iteration is much more expensive, and the Poisson solve (2.9) accounts for approximately 60% of wall clock time.

For domains with Dirichlet and Neumann boundaries, we extrapolate using values at the neighboring grid points and boundary data to fill ghost points. For some grid staggerings, the discrete second derivative operator approximates a nonunit multiple of its continuous counterpart near the boundary. To account for this, but maintain symmetry of the Helmholtz matrices, we scale equations involving near-boundary values, excluding the offending discrete second derivative and ghost cell terms. This error does not affect the Poisson solve (2.9). The correction recovers up to quadratic solutions exactly while retaining the use of PCG for the linear solves, but requires 3 additional diagonal matrix-vector multiplications per RK stage. For details, see Appendix A.

2.2 Cell energy models

In this section, we describe the various forms of energy density used in our simulations and give analytic expressions for each. We consider four kinds of energy densities: those corresponding to (damped) spring forces, tensions, dissipative forces, and Canham-Helfrich bending forces. The different cell types respond differently to deformation. We list the energy densities used to model each type of cell and the accompanying parameter values but defer discussion of force discretizations to Sections 3.2.2 and 3.2.3. Because our ultimate goal is a three-dimensional simulation, we limit our descriptions to the three-dimensional case. Considerations for the two-dimensional case are treated elsewhere [27].

We begin with Hookean and damped spring energy density, which is the simplest energy density we consider. It depends only on surface locations $\boldsymbol{X}$ and the surface velocity $\dot{\boldsymbol{X}}$, where the superposed dot denotes partial differentiation with respect to t, and takes the form

$$W_{\text{spring}}(\boldsymbol{X}, \dot{\boldsymbol{X}}) = \frac{k}{2}(\boldsymbol{X} - \boldsymbol{X}')^2 + \frac{\eta}{2}(\dot{\boldsymbol{X}} - \dot{\boldsymbol{X}}')^2. \tag{2.10}$$

where $\boldsymbol{X}'$ is the tether location, $\dot{\boldsymbol{X}}' = \partial \boldsymbol{X}'/\partial t$ is the prescribed velocity, k is the spring constant, and η is the damping constant. Due to the lack of information about the mechanical properties of endothelial cells, we model the endothelium as a rigid, stationary object with $k_{\text{endo}} = 2.5\,\text{dyn/cm}$ and $\eta_{\text{endo}} = 2.5 \times 10^{-7}\,\text{dyn} \cdot \text{s/cm}$, chosen to be as large as possible for the chosen spatial and temporal step size, and $\dot{\boldsymbol{X}}' = \boldsymbol{0}$. We compare different choices for $\boldsymbol{X}'$ in Chapter 5.

Next, we consider the tension densities for RBCs and platelets. These penalize stretching and areal dilation of the cell membranes. Let λ_1 and λ_2 be the principal extensions, *i.e.*, the maximal and minimal ratios of stretching relative to a reference configuration. We define the invariants $I_1 = \lambda_1^2 + \lambda_2^2 - 2$ and $I_2 = \lambda_1^2\lambda_2^2 - 1$, which measure relative changes in length and area, respectively, such that $I_1 = I_2 = 0$ correspond to a rigid body motion. We express the tension density in terms of these invariants. Skalak's Law was designed specifically for RBCs [45]:

$$W_{\text{Sk}}(I_1,\, I_2) = \frac{E}{4}\left(I_1^2 + 2I_1 - 2I_2\right) + \frac{G}{4}I_2^2. \tag{2.11}$$

E is the shear modulus, and G is the bulk modulus. The shear and bulk moduli for RBCs is estimated to be $E = 6 \times 10^{-3}\,\text{dyn/cm}$ and $G = 5 \times 10^{2}\,\text{dyn/cm}$ [37], but we follow Fai *et al.* and use $E_{\text{RBC}} = 2.5 \times 10^{-3}\,\text{dyn/cm}$ and $G_{\text{RBC}} = 2.5 \times 10^{-1}\,\text{dyn/cm}$ [30]. We use the shape given by Evans & Fung for the reference RBC with radius $R_0 = 3.91\,\mu\text{m}$ [28]:

$$\hat{\boldsymbol{X}}_{\text{RBC}}(\theta,\, \varphi) = R_0 \begin{bmatrix} \cos\theta\cos\varphi \\ \sin\theta\cos\varphi \\ z(\cos^2\varphi)\sin\varphi \end{bmatrix},$$

where $z(r) = 0.105 + r - 0.56r^2$. Platelets, on the other hand, do not have a purpose-built constitutive law, but are stiffer than RBCs. We use the neo-Hookean model

$$W_{\text{nH}}(I_1,\, I_2) = \frac{E}{2}\left(\frac{I_1 + 2}{\sqrt{I_2 + 1}} - 2\right) + \frac{G}{2}\left(\sqrt{I_2 + 1} - 1\right)^2 \tag{2.12}$$

with $E_{\text{plt}} = 1 \times 10^{-1}\,\text{dyn/cm}$ and $G_{\text{plt}} = 1\,\text{dyn/cm}$, and an ellipsoidal reference configuration [32]

$$\hat{\boldsymbol{X}}_{\text{plt}}(\theta,\, \varphi) = \begin{bmatrix} 1.55\,\mu\text{m}\cos\theta\cos\varphi \\ 1.55\,\mu\text{m}\sin\theta\cos\varphi \\ 0.5\,\mu\text{m}\sin\varphi \end{bmatrix}.$$

Platelets and RBCs also respond to changes in membrane curvature. Let H be the membrane's mean curvature. The Canham-Helfrich bending energy density takes the form [25]

$$W_{\text{CH}}(H) = 2\kappa(H - H')^2, \tag{2.13}$$

where κ is the bending modulus in units of energy, and H' is the spontaneous or *preferred* curvature. RBCs generate a relatively weak response to changes in curvature. Its bending modulus is estimated to be in the range $0.3\text{–}4 \times 10^{-12}\,\text{erg}$ [37]. We use a bending modulus of $\kappa_{\text{RBC}} = 2 \times 10^{-12}\,\text{erg}$ and a preferred curvature $H' = 0$ for RBCs. RBCs, therefore,

tend to locally flatten their membranes. For platelets, we use a larger bending modulus of $\kappa_{\text{plt}} = 2 \times 10^{-11}$ erg and a preference for its reference curvature. Together with the neo-Hookean tension above, this maintains a fairly rigid platelet.

Finally, we consider dissipative energy, which causes the membrane to exhibit a viscoelastic response to strain. It takes the form [42]

$$W_{\text{dissip}}(\dot{\lambda}_1, \dot{\lambda}_2) = \frac{\nu}{2}\left(\frac{\dot{\lambda}_1^2}{\lambda_1^2} + \frac{\dot{\lambda}_2^2}{\lambda_2^2}\right), \tag{2.14}$$

where ν is the membrane viscosity, and $\dot{\lambda}_i$ is the rate of change of λ_i. We imbue only the RBC with viscoelasticity. We find this effective in eliminating some numerical instabilities. While Evans and Hochmuth suggest a viscosity of approximately 1×10^{-3} dyn $\cdot$ s/cm [29], we find this to be prohibitively expensive in practice, due to time step restrictions, and instead use $\nu_{\text{RBC}} = 2.5 \times 10^{-7}$ dyn $\cdot$ s/cm.

2.3 Geometry of reconstructed surfaces

Let Γ denote the membrane of an RBC, platelet, or the endothelium at time t. To discretize Γ, we choose a fixed set of material coordinates. Throughout a simulation, we track their corresponding Cartesian locations. From these points, we approximately reconstruct the entire cell membrane. We aim to construct a sufficiently smooth reconstruction from which we approximate surface forces. The tool of choice is radial basis function interpolation.

2.3.1 Interpolation with radial basis functions

Radial basis functions (RBFs) are a meshfree approach to scattered data approximation where structural information is encoded purely as pointwise distances. With some exceptions, they are an appropriate tool for interpolation at arbitrary locations. This contrasts with, *e.g.*, polynomials, where points must be chosen at grid vertices, or spherical harmonics, for which special node sets are typically used. RBFs are a viable approach for representing cells on par with Fourier methods [43]. They are therefore appealing for representing blood cells.

Because our choice of force model for the endothelium does not require geometric information, we limit the discussion to RBCs and platelets, which are topologically spherical. The 2-sphere, $\mathbb{S}^2$, has parametrization

$$\chi(\theta, \varphi) = \begin{bmatrix} \cos\theta\cos\varphi \\ \sin\theta\cos\varphi \\ \sin\varphi \end{bmatrix},$$

$(\theta, \varphi) \in (-\pi, \pi] \times [-\pi/2, \pi/2]$. Let $\Theta^{\mathrm{d}} = \{(\theta_i, \varphi_i)\}$ be a set of n_d distinct *data sites*, and $\chi_i^{\mathrm{d}} = \chi(\theta_i, \varphi_i)$ for each $(\theta_i, \varphi_i) \in \Theta^{\mathrm{d}}$. Suppose we wish to approximate $\psi(\chi)$, defined on $\mathbb{S}^2$. From *basic function* ϕ, we form our interpolatory basis with the RBFs $\phi(\|\chi - \chi_i^{\mathrm{d}}\|)$. Attractive choices for ϕ are the polyharmonic splines (PHS),

$$\text{PHS:} \quad \phi(r) = \begin{cases} r^{2k} \log r, \\ r^{2k+1}, \end{cases} \qquad \text{for } k \in \mathbb{N},$$

which do not require a shape parameter, unlike Gaussian or multiquadric kernels [31]. However, PHS are finitely differentiable and conditionally positive definite; they require additional polynomial terms up to degree k to guarantee a unique interpolant. Heuristically, n_d is chosen so that data sites outnumber polynomials at least 2-to-1 to maintain reasonable conditioning. On $\mathbb{S}^2$, the polynomials are typically spherical harmonics. We denote the polynomials by $p_k(\chi)$, $k = 1, \ldots, n_p$. The interpolant takes the form

$$s(\chi) = \sum_{i=1}^{n_d} c_i \phi(\|\chi - \chi_i^{\mathrm{d}}\|) + \sum_{k=1}^{n_p} d_k p_k(\chi), \tag{2.15}$$

and exactly recovers ψ at each of the data sites, $s(\chi_j^{\mathrm{d}}) = \psi(\chi_i^{\mathrm{d}})$, for $j = 1, \ldots, n_d$. We further constrain c_i so that the polynomials recover polynomial data,

$$\sum_{i=1}^{n_d} c_i p_k(\chi_i^{\mathrm{d}}) = 0. \tag{2.16}$$

Collecting the values $\Phi = (\phi(\|\chi_j^{\mathrm{d}} - \chi_i^{\mathrm{d}}\|))$, $P = (p_k(\chi_j^{\mathrm{d}}))$, $\boldsymbol{c} = (c_i)$, $\boldsymbol{d} = (d_k)$, and $\boldsymbol{\psi} = (\psi(\chi_j^{\mathrm{d}}))$, we form the dense symmetric block system

$$\begin{bmatrix} \Phi & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{d} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\psi} \\ \boldsymbol{0} \end{bmatrix}, \tag{2.17}$$

where the matrix block 0 is the $n_p \times n_p$ zero matrix and $\boldsymbol{0}$ is a vector of n_p zeros. Because Θ^{d} is fixed, we need only construct this matrix once.

For RBC and platelet parametrizations, we identify the point $\boldsymbol{X}(\theta, \varphi, t)$ on Γ with the point $\chi(\theta, \varphi)$ on $\mathbb{S}^2$. Each component of $\boldsymbol{X}$ is a function defined on $\mathbb{S}^2$. By sampling $\boldsymbol{X}$ at each point in Θ^{d}, we can approximately reconstruct the surface by interpolating each of the components. It is clear from Equations (2.11)–(2.14) that computing the force density (2.4) requires values of I_1, I_2, and H, among others. These values are derived from the first and second derivatives of $\boldsymbol{X}$. To meet the smoothness requirements to evaluate force densities, we use $\phi(r) = r^7$ for each cell, with up to 5^{th} order spherical harmonics for RBCs and just the constant polynomial for platelets. While this does not guarantee a unique interpolant for the platelet, the resulting system (2.17) is invertible nonetheless. The interpolants are then thrice differentiable. We now need the appropriate discrete differential operators.

2.3.2 Discrete linear surface operators

Let $\mathcal{L}$ be a linear operator. In particular, we are interested in the first- and second-order partial differential operators, $\partial/\partial\theta$, $\partial^2/\partial\theta\partial\varphi$, *etc*. We approximate $\mathcal{L}\psi$ by applying $\mathcal{L}$ analytically to s. This is straightforward, given a parametrized metric. For $\mathbb{S}^2$, this is

$$\begin{aligned}\|\boldsymbol{\chi}(\theta_j,\,\varphi_j) - \boldsymbol{\chi}(\theta_i,\,\varphi_i)\| &= \sqrt{2(1 - \cos\varphi_j\cos\varphi_i\cos(\theta_j - \theta_i) - \sin\varphi_j\sin\varphi_i)} \\ &= \sqrt{2(1 - \boldsymbol{\chi}(\theta_j,\,\varphi_j)\cdot\boldsymbol{\chi}(\theta_i,\,\varphi_i))}.\end{aligned} \tag{2.18}$$

However, evaluating $\mathcal{L}s$ at each data site involves dense operations against a $n_d \times (n_d + n_p)$ matrix. Depending on the needs of the simulation, the number of data sites may be large. In the interest of saving memory and time for such cases, we opt instead to use fewer data sites to reconstruct the surface, and choose a larger set of n_s *sample sites*, Θ^{s}. We must then also consider $\mathcal{L}$ the identity operator in order to obtain $\boldsymbol{X}$ at sample sites. Evaluating $\mathcal{L}s$ at each sample site, $\boldsymbol{\chi}_j^{\mathrm{s}} = \boldsymbol{\chi}(\theta_j,\,\varphi_j)$ for each $(\theta_j,\,\varphi_j) \in \Theta^{\mathrm{s}}$, and collecting values $\mathcal{L}\Phi = (\mathcal{L}\phi(\|\boldsymbol{\chi} - \boldsymbol{\chi}_i^{\mathrm{d}}\|)|_{\boldsymbol{\chi}=\boldsymbol{\chi}_j^{\mathrm{s}}})$ and $\mathcal{L}P = (\mathcal{L}p_k(\boldsymbol{\chi})|_{\boldsymbol{\chi}=\boldsymbol{\chi}_j^{\mathrm{s}}})$, we have

$$\begin{aligned}\begin{bmatrix} \mathcal{L}\Phi & \mathcal{L}P \end{bmatrix}\begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{d} \end{bmatrix} &= \begin{bmatrix} \mathcal{L}\Phi & \mathcal{L}P \end{bmatrix}\begin{bmatrix} \Phi & P \\ P^T & 0 \end{bmatrix}^{-1}\begin{bmatrix} \boldsymbol{\psi} \\ \boldsymbol{0} \end{bmatrix} \\ &:= \begin{bmatrix} L & * \end{bmatrix}\begin{bmatrix} \boldsymbol{\psi} \\ \boldsymbol{0} \end{bmatrix},\end{aligned} \tag{2.19}$$

where we have used Equation (2.17) to substitute for $\boldsymbol{c}$ and $\boldsymbol{d}$. The matrix L is the discrete analogue of $\mathcal{L}$ applied at each sample site. The block marked by $*$ is multiplied by zeros, and can be discarded. We perform this procedure for each first and second derivative operator, and for the identity operator if $\Theta^{\mathrm{d}} \neq \Theta^{\mathrm{s}}$. The discrete identity operator evaluates the interpolant at sample sites to obtain locations $\boldsymbol{X}_j^{\mathrm{s}}$, at which we compute force density. When we do not need to make a distinction, we use the notation Γ_h to represent the set of Cartesian locations corresponding to every data or sample site, and n_γ to represent the appropriate cardinality. For fixed Θ^{d} and Θ^{s}, we construct these operators only once. The `cuSOLVER` library provides a parallel `LAPACK`-like interface to solve the linear systems on the GPU [39].

The quantities I_1, I_2, and H in Section 2.2 are calculated using local geometric data. Their computation is trivially parallelizable given approximations to the first and second derivatives of $\boldsymbol{X}$ with respect to material coordinates. Application of the dense discrete differential operators is also performed in parallel with a parallel implementation of `BLAS`. NVIDIA® provides `cuBLAS` for this purpose [39]. To compute a force from a force density, we need to approximate quadrature weights, or surface patch areas, for each sample site.

2.3.3 Radial basis function-based quadrature

We now use the known parametrization of $\mathbb{S}^2$ to compute RBF-based quadrature weights on $\mathbb{S}^2$ as a preliminary step in computing quadrature weights on any surface diffeomorphic to $\mathbb{S}^2$, namely, RBC and platelet membranes.

As before, consider a function $\psi(\boldsymbol{\chi}) : \mathbb{S}^2 \to \mathbb{R}$. We wish to find a set of quadrature weights ω_j such that

$$\int_{\mathbb{S}^2} \psi(\boldsymbol{\chi}) \, \mathrm{d}\boldsymbol{\chi} \approx \sum_{j=1}^{n_s} \omega_j \psi(\boldsymbol{\chi}_j^{\mathrm{s}}).$$

We use a variant of the technique described by Fuselier *et al.* [33]. Choosing $\psi(\boldsymbol{\chi}) = \phi(\|\boldsymbol{\chi} - \boldsymbol{\chi}_i^{\mathrm{s}}\|)$ for each $\boldsymbol{\chi}_i^{\mathrm{s}}$ we see that

$$\sum_{j=1}^{n_s} \omega_j \phi(\|\boldsymbol{\chi}_j^{\mathrm{s}} - \boldsymbol{\chi}_i^{\mathrm{s}}\|) \approx \int_{\mathbb{S}^2} \phi(\|\boldsymbol{\chi} - \boldsymbol{\chi}_i^{\mathrm{s}}\|) \, \mathrm{d}\boldsymbol{\chi} := \mathcal{L}\phi|_{\boldsymbol{\chi}=\boldsymbol{\chi}_i^{\mathrm{s}}}. \tag{2.20}$$

However, because the spherical metric (2.18) depends only on the angle between two points, $\mathbb{S}^2$ is homogeneous. As a consequence, $\mathcal{L}\phi$ is constant over the sphere. We therefore expect the left hand side to be a constant, which we denote $-I_\phi$. We require further that ω_j sum to the surface area of $\mathbb{S}^2$, *i.e.*,

$$\sum_{j=1}^{n_s} \omega_j = 4\pi. \tag{2.21}$$

Treating I_ϕ as an unknown scalar, we rewrite the constraints (2.20) and (2.21) as a symmetric block linear system for ω_j. Let $\Phi = (\phi(\|\boldsymbol{\chi}_j^{\mathrm{s}} - \boldsymbol{\chi}_i^{\mathrm{s}}\|))$, $\boldsymbol{\omega} = (\omega_j)$, $\mathbf{0}$ be a vector of n_s zeros, and $\mathbf{1}$ be defined similarly with ones. Then

$$\begin{bmatrix} \Phi & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega} \\ I_\phi \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 4\pi \end{bmatrix}. \tag{2.22}$$

I_ϕ serves as a Lagrange multiplier that enforces (2.21), and $-I_\phi$ is a good approximation to $\mathcal{L}\phi$. By choosing $\phi(r) = r$, we guarantee a unique solution with weights that converge at 3^{rd} order. It is possible to improve the order of the quadrature weights by increasing the order of the PHS at the potential cost of poorer conditioning and either loss of invertibility or requiring knowledge of higher-order moments [33]. For each set of quasiuniform points on $\mathbb{S}^2$ we tested, the system is invertible for $\phi(r) = r^{2k+1}$ for $1 \leq k \leq 4$, but quality of the quadrature weights deteriorates with increased number of sample sites or higher-order PHS, typically becoming unusable around $k = 4$. When higher-order moments are used to improve the order, poor conditioning typically causes the approximation to $\mathcal{L}\phi$ to become nonconstant. From quadrature weights and geometric information for the cell surface, we

can compute quadrature weights for RBCs and platelets, We complete this calculation in Section 3.2.4.2.

The methods described in this section are not restricted to the sphere. In Chapter 5, we simulate the endothelium, which is a topological torus, due to the periodicity of the domain. Though we do not need geometric information for the force model applied to the endothelium, the RBF methods above are also applicable to the torus, and therefore the endothelium. We refer the reader to Appendix B for details. We use a modification of these methods as part of the simulation initialization process, described in Section 5.2.1.

Fluid solver and surface representations at hand, we need a way to allow the fluid and cells to interact. We achieve this through the immersed boundary method, which we detail in the next section.

2.4 The immersed boundary method

To simulate whole blood, we use the immersed boundary (IB) method. The IB method was developed by Peskin to study the flow of blood around heart valves [40]. It has since been used to simulate, among numerous others, vibrations in the inner ear [23], the opening of a porous parachute [36], and sperm motility [26], and has generated numerous related methods. The use of RBF-based methods to represent immersed structures within the IB method has been used to simulate flow around an aggregate of platelets, and bears the moniker RBF-IB [44]. The IB method remains popular for modeling fluid-structure interaction because of its simplicity. By treating the immersed structures as an extension of the fluid, an arbitrary parcel in Ω is assumed to behave like a fluid, so its motion is governed by the incompressible Navier-Stokes equations (2.1) and (2.2), whether the parcel contains a portion of a membrane or not. This allows for the use of the MAC grid from Section 2.1 without modification. As a further simplification, we will ignore the difference in viscosity between the plasma and the interior of the blood cells.

Γ is assumed impermeable to the fluid and moves at the local fluid velocity. Analytically, the fluid velocity can be evaluated at surface point $\boldsymbol{X}$ by integrating $\boldsymbol{u}(\boldsymbol{x}, t)$ against the translated Dirac delta function, $\delta(\boldsymbol{x} - \boldsymbol{X})$. However, the components of the fluid velocity are only known at a fixed set of points, and different components at different sets of points. It is extremely unlikely that $\boldsymbol{X}$ would ever coincide with an Eulerian grid point, and even so, we could only interpolate one component of the velocity. When discretized, the IB method replaces the singular Dirac delta function with a smoothed, h-dependent analogue, δ_h. The

Lagrangian point $\boldsymbol{X}$ then evolves according to

$$\begin{aligned} \boldsymbol{e}_a \cdot \dot{\boldsymbol{X}} &= h^3 \sum_{i=1}^{n_\omega^a} \boldsymbol{e}_a \cdot \boldsymbol{u}(\boldsymbol{x}_i)\delta_h(\boldsymbol{x}_i - \boldsymbol{X}) \\ &\approx \int_\Omega \boldsymbol{e}_a \cdot \boldsymbol{u}(\boldsymbol{x})\delta(\boldsymbol{x} - \boldsymbol{X})\,\mathrm{d}\boldsymbol{x}, \end{aligned} \tag{2.23}$$

where i enumerates the points in Ω_h^a. Because we discretize each component of $\boldsymbol{u}$ at different spatial locations, it necessary to consider each component individually, which we write as a dot product with canonical basis vector $\boldsymbol{e}_a$ for $a = 1, \ldots, d$. As a boundary deforms, it generates a force density $\boldsymbol{F} = \boldsymbol{F}(\theta, \varphi, t)$, which it imparts onto the fluid as $\boldsymbol{f}$ in (2.1). By similar reasoning as the velocity, $\boldsymbol{F}$ is transferred to the fluid at $\boldsymbol{x}$ via

$$\begin{aligned} \boldsymbol{e}_a \cdot \boldsymbol{f}(\boldsymbol{x}) &= \sum_{j=1}^{n_\gamma} \Delta A_j \boldsymbol{e}_a \boldsymbol{F}(\boldsymbol{X}_j)\delta_h(\boldsymbol{x} - \boldsymbol{X}_j) \\ &\approx \int_\Gamma \boldsymbol{e}_a \cdot \boldsymbol{F}(\boldsymbol{X})\delta(\boldsymbol{x} - \boldsymbol{X})\,\mathrm{d}\boldsymbol{X}, \end{aligned} \tag{2.24}$$

where j enumerates Lagrangian grid points, and ΔA_j is the integration weight corresponding to Lagrangian grid point $\boldsymbol{X}_j$. By evaluating Equation (2.24) at each point in Ω_h^a, we can gather the values of $\delta_h(\boldsymbol{x}_i - \boldsymbol{X}_j)$, with $\boldsymbol{X}_j$ the value of $\boldsymbol{X}$ evaluated at the j^{th} data site, into a spreading matrix, $\mathcal{S}$. The values used in interpolation are similar, but $\boldsymbol{X}$ is now sampled at sample sites. We denote the interpolation matrix $\mathcal{I}$. Traditionally, forces are spread from, and velocities interpolated to, the same set of points. This results in adjoint spreading and interpolation operators and guarantees conservation of energy. The interface points are heuristically chosen to be within h of one another as a means of preventing leakage across an interface. We choose the sample sites to satisfy this condition, but fewer data sites, which still provides a good surface reconstruction [44]. As a result, surface points corresponding to sample sites may not move at the local fluid velocity, but this alone may not be problematic [34].

Equations (2.23) and (2.24) are the glue that couple the Eulerian components, (2.1) and (2.2) to the Lagrangian ones, (2.4). Collectively, these comprise a single step of the IB method. At time $t = nk$, we

(a) interpolate $\boldsymbol{u}^n$ to $\boldsymbol{X}^n$ to get $\dot{\boldsymbol{X}}^n$,

(b) predict new Lagrangian locations, $\boldsymbol{X}^* = \boldsymbol{X}^n + \Delta t \dot{\boldsymbol{X}}^n$,

(c) compute Lagrangian force densities, $\boldsymbol{F}^{n+1}$,

(d) spread $\boldsymbol{F}^{n+1}$ to $\boldsymbol{X}^*$ to get $\boldsymbol{f}^{n+1}$,

(e) solve for fluid velocity $\boldsymbol{u}^{n+1}$,

(f) interpolate $\boldsymbol{u}^{n+1}$ to $\boldsymbol{X}^n$ to get $\dot{\boldsymbol{X}}^{n+1}$, and

(g) update Lagrangian locations, $\boldsymbol{X}^{n+1} = \boldsymbol{X}^n + \Delta t \dot{\boldsymbol{X}}^{n+1}$,

with some steps modified or repeated, depending on the timestepping scheme. We have already described (e) in Section 2.1. The updates (b) and (g) are simple applications of the `axpy` routine from `BLAS`. Chapter 3 bridges the gap between our interpolated surfaces and the surface forces for (c). Chapter 4 provides parallelization strategies for the remaining steps. Finally, the pieces are put together to simulate blood in Chapter 5.

References

[21] M. F. Adams, M. Brezina, J. Hu, and R. Tuminaro, *Parallel multigrid smoothing: polynomial versus gauss–seidel*, Journal of Computational Physics, 188 (2003), pp. 593–610.

[22] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, *Implicit-explicit runge-kutta methods for time-dependent partial differential equations*, Applied Numerical Mathematics, 25 (1997), pp. 151–167.

[23] R. P. Beyer Jr, *A computational model of the cochlea using the immersed boundary method*, Journal of Computational Physics, 98 (1990), pp. 145–162.

[24] D. L. Brown, R. Cortez, and M. L. Minion, *Accurate projection methods for the incompressible navier–stokes equations*, Journal of Computational Physics, 168 (2001), pp. 464–499.

[25] P. B. Canham, *The minimum energy of bending as a possible explanation of the biconcave shape of the human red blood cell*, Journal of Theoretical Biology, 26 (1970), pp. 61–81.

[26] R. Dillon and J. Zhuo, *Using the immersed boundary method to model complex fluid/structure interaction in sperm motility*, Discrete and Continuous Dynamical Systems Series B, 15 (2011), pp. 343–355.

[27] L. C. Erickson, *Blood flow dynamics: a lattice Boltzmann immersed boundary approach*, PhD thesis, University of Utah, Salt Lake City, Dec. 2010.

[28] E. A. Evans and Y.-C. Fung, *Improved measurements of erythrocyte geometry*, Microvascular Research, 4 (1972), pp. 335–347.

[29] E. A. Evans and R. M. Hochmuth, *Membrane viscoelasticity*, Biophysical Journal, 16 (1976), pp. 1–11.

[30] T. G. Fai, B. E. Griffith, Y. Mori, and C. S. Peskin, *Immersed boundary method for variable viscosity and variable density problems using fast constant-coefficient linear solvers I: numerical method and results*, SIAM Journal on Scientific Computing, 35 (2013), pp. B1132–B1161.

[31] G. E. Fasshauer, *Meshfree approximation methods with MATLAB*, World Scientific, Jan. 2007.

[32] M. M. Frojmovic and J. G. Milton, *Human platelet size, shape, and related functions in health and disease*, Physiological Reviews, 62 (1982), pp. 185–261.

[33] E. J. Fuselier, T. Hangelbroek, F. J. Narcowich, J. D. Ward, and G. B. Wright, *Kernel based quadrature on spheres and other homogeneous spaces*, Numerische Mathematik, 127 (2013), pp. 57–92.

[34] B. E. Griffith and X. Luo, *Hybrid finite difference/finite element immersed boundary method*, International Journal for Numerical Methods in Biomedical Engineering, 33 (2017), p. e2888.

[35] F. H. Harlow and J. E. Welch, *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, Physics of Fluids, 8 (1965), pp. 2182–2189.

[36] Y. Kim and C. S. Peskin, *2-D parachute simulation by the immersed boundary method*, SIAM Journal on Scientific Computing, 28 (2006), pp. 2294–2312.

[37] N. Mohandas and E. A. Evans, *Mechanical properties of the red cell membrane in relation to molecular structure and genetic defects*, Annual Review of Fluid Mechanics, 23 (1994), pp. 787–818.

[38] Y. Morinishi, T. S. Lund, O. V. Vasilyev, and P. Moin, *Fully conservative higher order finite difference schemes for incompressible flow*, Journal of Computational Physics, 143 (1998), pp. 90–124.

[39] NVIDIA, P. Vingelmann, and F. H. Fitzek, *CUDA*, 2021, https://developer.nvidia.com/cuda-toolkit.

[40] C. S. Peskin, *Flow patterns around heart valves: a digital computer method for solving the equations of motion*, PhD thesis, Yeshiva University, New York, 1972.

[41] C. S. Peskin, *The immersed boundary method*, Acta Numerica, 11 (2002), pp. 479–517.

[42] P. Rangamani, A. Agrawal, K. K. Mandadapu, G. Oster, and D. J. Steigmann, *Interaction between surface shape and intra-surface viscous flow on lipid membranes*, Biomechanics and Modeling in Mechanobiology, 12 (2012), pp. 833–845.

[43] V. Shankar, G. B. Wright, A. L. Fogelson, and R. M. Kirby, *A study of different modeling choices for simulating platelets within the immersed boundary method*, Applied Numerical Mathematics, 63 (2013), pp. 58–77.

[44] V. Shankar, G. B. Wright, R. M. Kirby, and A. L. Fogelson, *Augmenting the immersed boundary method with radial basis functions (RBFs) for the modeling of platelets in hemodynamic flows*, International Journal for Numerical Methods in Fluids, 79 (2015), pp. 536–557.

[45] R. Skalak, A. Tozeren, R. P. Zarda, and S. Chien, *Strain energy function of red blood cell membranes*, Biophysical Journal, 13 (1973), pp. 245–264.

CHAPTER 3

A CONTINUOUS ENERGY-BASED IMMERSED BOUNDARY METHOD FOR ELASTIC SHELLS

In this chapter, we present a method for computing surface forces for smooth global interpolants. We use the term *continuous energy* in the sense that the interpolation scheme allows us to evaluate surface energy, or a force derived from it, at an arbitrary location on the surface. In contrast, discrete methods define energy and forces only at fixed locations.

The paper presented in this chapter appears in *Journal of Computational Physics* **371** (2018) 333–362. It is reprinted with permission from the publisher. Here, we use spherical harmonics to construct the cell surface representations. These methods are analogous to those in the previous chapter, where we describe them in the context of RBFs. The additional details presented here, for analytically computing force densities, surface forces densities, and surface quadrature weights, are equally applicable to RBF-based surface representations.

Journal of Computational Physics 371 (2018) 333–362

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A continuous energy-based immersed boundary method for elastic shells

Ondrej Maxian [a,b], Andrew T. Kassen [c], Wanda Strychalski [b,*]

[a] Department of Chemical and Biomolecular Engineering, Case Western Reserve University, Cleveland, OH 44106, the United States of America
[b] Department of Mathematics, Applied Mathematics, and Statistics, Case Western Reserve University, Cleveland, OH 44106, the United States of America
[c] Department of Mathematics, University of Utah, UT 84112, the United States of America

ARTICLE INFO

Article history:
Received 21 November 2017
Received in revised form 28 April 2018
Accepted 27 May 2018
Available online 29 May 2018

Keywords:
Fluid–structure interaction
Stokes flow
Hyperelasticity
Red blood cells
Blebbing

ABSTRACT

The immersed boundary method is a mathematical formulation and numerical method for solving fluid–structure interaction problems. For many biological problems, such as models that include the cell membrane, the immersed structure is a two-dimensional infinitely thin elastic shell immersed in an incompressible viscous fluid. When the shell is modeled as a hyperelastic material, forces can be computed by taking the variational derivative of an energy density functional. A new method for computing a continuous force function on the entire surface of the shell is presented here. The new method is compared to a previous formulation where the surface and energy functional are discretized before forces are computed. For the case of Stokes flow, a method for computing quadrature weights is provided to ensure the integral of the elastic spread force density remains zero throughout a dynamic simulation. Tests on the method are conducted and show that it yields more accurate force computations than previous formulations as well as more accurate geometric information such as mean curvature. The method is then applied to a model of a red blood cell in capillary flow and a 3D model of cellular blebbing.

1. Introduction

The immersed boundary (IB) method was first introduced by Charles Peskin in 1972 to model blood flow around heart valves [1]. Since then, it has been applied to a wide variety of biological models, including insect flight [2,3], animal swimming [4], and cellular mechanics [5]. In three-dimensional models, computing the elasticity of an immersed 2D elastic membrane or 3D elastic solid as it deforms is more challenging than in 2D because it involves stress tensors. The first studies in this area used lattice-spring models of elasticity (e.g. [6]), but these methods are disconnected from a constitutive law. The finite element method has also been used to compute the elasticity of immersed structures [7–10], but has not been extended to infinitely thin shells. The IB method has been applied to hyperelastic solids or shells whose energies are governed by a strain energy density functional [11,12]. However, the surface or solid representation has been assumed to be piecewise linear, and the accuracy of force computations with such a method has not been rigorously tested.

The authors showed in [11] that for a hyperelastic solid, forces can be computed without using stress tensors. In this work, the derivation was limited to Cartesian coordinates and applied to solids and thick elastic shells. The method was

* Corresponding author.
E-mail addresses: oxm60@case.edu (O. Maxian), kassen@math.utah.edu (A.T. Kassen), wis6@case.edu (W. Strychalski).

https://doi.org/10.1016/j.jcp.2018.05.045
0021-9991/

later applied to infinitely thin elastic shells represented in two curvilinear coordinates and was subsequently used to model red blood cells [12], osmotic swelling [13], and two-phase gels [14]. However, these models have used a piecewise linear boundary representation (surface triangulation) [12] or marker and cell discretization of the surface [14]. Both approaches are based on the assumption that curved surfaces are locally planar, which introduces surface discretization error and fails to capture the inherent curvature of the surface.

Studies on surface representations have generally been conducted separately from those on force computations. In [15, 16], the authors focus on representing thin surfaces continuously with spherical harmonic or radial basis function (RBF) interpolants. In both cases, force computations are treated through simple explicit expressions for surface tension [15] or fiber elasticity [16]. The authors do not consider finding forces through variational derivatives.

Boundary integral methods (BIMs) have been progressing concurrently with the IB method and have been used by others to model fluid–structure interactions in zero Reynolds number flow [17–20]. In BI methods, the "hydrodynamic traction jump" across the membrane [17], which is equivalent to the Lagrangian elastic force density per unit current area [20], must be computed independently prior to integration. [21] and [22] both demonstrate the use of a spectrally convergent spherical harmonic representation for surfaces that allows for the calculation of force in BIMs. However, the computations in [21] and [22] both follow the traditional formulation of BI methods in computing the traction jump from a system of equations involving the Cauchy stress tensor, which itself comes from a series of tensor expansions. Additionally, BI methods require numerical schemes to resolve integrals that are singular at the immersed surfaces.

The aim of this work is to provide a "bridge" between the fields of surface representation and energy-based force functions within the IB method. Our goal is to use the continuous surface representations from [15] to formulate a continuous function that represents the force density on the membrane as a function of its curvilinear coordinates. In this manner, we avoid the discretization error from surface representation. The advantages of the approach presented here are that the force is computed directly in a single computation, bypassing the need to compute stress tensors at each time step, the interpolation of structure velocities in the IB method occurs over a coarser mesh, and the use of the IB method avoids the need for numerical methods to compute singular integrals as in BI methods.

Since many biological phenomena occur on the cellular level, the small length scales (microns) lead to very small Reynolds numbers (Re $\sim 10^{-5}$–10^{-2}). In order to simulate Stokes equations with periodic boundary conditions, the integral of the force density over the fluid domain must be zero [23]. In [24], the authors proposed a method for ensuring this condition is met when tether forces are used in the IB method. For translation invariant hyperelastic materials represented by continuous functions, the integral condition is satisfied in the continuous formulation [25]. However, extra care must be taken to ensure that the discrete integral of the forces in the IB method is zero. Another feature of this work is that we provide a method for computing the correct quadrature weights for closed surfaces so that the IB method can be used with 3D Stokes flow.

The rest of the paper is organized as follows. In Section 2, we present the mathematical formulation of the immersed membrane problem, including our method of computing force densities and correct area quadrature weights to ensure that the forces on the fluid integrate to zero in Stokes' flow. We continue by describing an alternative method based on piecewise linear surface discretization, which we use for comparison. In Section 3, we discuss the numerical formulation and discretization of the problem and provide implementation details. We conclude by presenting our results in Section 4, where we perform force computations and IB simulations on two test objects. In Section 5, we apply our model to two different biological processes: a red blood cell flowing through a capillary and a 3D model of cellular blebbing.

2. Mathematical formulation

In this section, we begin by presenting an overview of the model equations and mathematical formulation. We then describe how a continuous elastic force density function can be computed on shells using the energy-based formulation. We provide a method for computing quadrature weights so that the forces applied to the fluid integrate to zero. We compare this to the discretized surface formulation from [12,26] and conclude by presenting the energy functionals used in our numerical tests.

2.1. Immersed boundary method formulation

Our model system is an infinitely thin elastic shell immersed in a viscous fluid. For our applications, the shell represents the cell membrane. Due to the small length scales on the cellular level, inertial forces can be neglected and the fluid obeys Stokes' equations,

$$\mu\Delta\boldsymbol{u} - \nabla p + \boldsymbol{f} = \boldsymbol{0}, \tag{1}$$

$$\nabla\cdot\boldsymbol{u} = 0, \tag{2}$$

where $\boldsymbol{u}$ represents the fluid velocity, p is the pressure, $\boldsymbol{f}$ is an external force, and μ is the fluid viscosity. The model is formulated using the IB method so that the thin elastic shell is represented on a moving Lagrangian coordinate system while the velocity and pressure are represented on a fixed Eulerian grid. Elastic forces are computed on the Lagrangian grid and spread onto the Eulerian grid to construct the external force density. The spreading operator S is defined as

$$\boldsymbol{f} = \mathrm{S}\boldsymbol{F} = \int_{\Gamma} \boldsymbol{F}(\boldsymbol{q},t)\delta(\boldsymbol{x} - \boldsymbol{X}(\boldsymbol{q},t))\, d\boldsymbol{q}, \tag{3}$$

where Γ represents the Lagrangian structure (shell), $\boldsymbol{q} = (q_1, q_2)$ represents material surface coordinates, $\boldsymbol{X}(\boldsymbol{q},t)$ represents the position of the structure at time t, and $\delta(\boldsymbol{x})$ represents the Dirac delta function. We use the convention that lower case letters indicate quantities on the Eulerian grid and capitalized letters denote quantities on the Lagrangian structure. The structure is updated with the local fluid velocity, satisfying the no-slip boundary condition,

$$\frac{d\boldsymbol{X}}{dt} = \boldsymbol{U}, \tag{4}$$

where $\boldsymbol{U}$ is the interpolated fluid velocity. The interpolation operator is the adjoint of the spreading operator in Eq. (3) and is given by

$$\boldsymbol{U} = \mathrm{S}^*\boldsymbol{u} = \int_{\Omega} \boldsymbol{u}(\boldsymbol{x},t)\delta(\boldsymbol{x} - \boldsymbol{X}(\boldsymbol{q},t))\, d\boldsymbol{x}, \tag{5}$$

where Ω denotes the fluid domain.

2.2. Computing the elastic forces

We outline two different methods to compute Lagrangian forces analytically based on the assumption that the structure is a hyperelastic material characterized by an energy density functional

$$\mathcal{E} = \int_{\Gamma_0} W\, d\boldsymbol{q}, \tag{6}$$

where W is a strain energy density specified by a constitutive law, such as the neo-Hookean model. Exact forms of W are provided in Section 2.6. The energy-based model for computing Lagrangian forces on an immersed surface was derived in [11]. Let the surface Γ with undeformed configuration Γ_0 be defined by Lagrangian coordinates q_1 and q_2 (q_i's are the parameters). Then the force density per unit reference configuration is related to the variational derivative of energy $\frac{\delta\mathcal{E}}{\delta\boldsymbol{X}}$ by

$$\boldsymbol{F} = -\frac{\delta\mathcal{E}}{\delta\boldsymbol{X}}(q_1, q_2, t), \tag{7}$$

where $\mathcal{E}$ is the surface energy and t is time.

The variational derivative is defined as follows. Let $\phi_{\delta\boldsymbol{X}}(\alpha) = \mathcal{E}(\boldsymbol{X} + \alpha\delta\boldsymbol{X})$. Then the variational derivative is the function $\frac{\delta\mathcal{E}}{\delta\boldsymbol{X}}$ that obeys

$$\left.\frac{d\phi_{\delta\boldsymbol{X}}(\alpha)}{d\alpha}\right|_{\alpha=0} = \int_{\Gamma_0} \frac{\delta\mathcal{E}}{\delta\boldsymbol{X}}(q_1, q_2, t) \cdot \delta\boldsymbol{X}(q_1, q_2)\, dA_0. \tag{8}$$

In [11], this theory was applied to discrete solids in Cartesian coordinates. We extend the theory further to curvilinear coordinates on *continuous closed* surfaces.

In the subsequent sections, we describe two different approaches for computing the force density in Eq. (7). We first outline a procedure to find the variational derivative of the continuous energy density functional and follow with a discussion on computing the proper area weights to obtain force from force density. We next discuss the second method, the linear discrete surface method (LDSM) from [12]. The difference between the two methods lies primarily in the fact that our formulation relies on a *continuous* force function whose integral is discretized after forces are computed, whereas in LDSM, the surface and energy functional are first discretized. Forces are then computed after the discretized surface energy is exactly integrated.

2.3. Computing forces from the variational derivative of the energy functional

Since the surfaces in our applications are topological spheres, we parameterize our surface in terms of spherical coordinates. However, our derivations can be generalized to any coordinate system. We begin by defining a two dimensional surface in terms of spatial coordinates $q_1 = \lambda$ and $q_2 = \theta$. Let

$$\boldsymbol{Z}(\lambda,\theta) = \begin{pmatrix} \hat{X}(\lambda,\theta) \\ \hat{Y}(\lambda,\theta) \\ \hat{Z}(\lambda,\theta) \end{pmatrix} \text{ and } \boldsymbol{X}(\lambda,\theta) = \begin{pmatrix} X(\lambda,\theta) \\ Y(\lambda,\theta) \\ Z(\lambda,\theta) \end{pmatrix} \tag{9}$$

be the reference and current configurations of the surface, respectively, where $-\pi < \lambda \leq \pi$ and $-\pi/2 < \theta \leq \pi/2$. Here $\boldsymbol{X}, \boldsymbol{Z} : (\lambda, \theta) \mapsto \mathbb{R}^3$, and we have intentionally expressed Eq. (9) in a general form to emphasize that our derivation can be applied to any particular choice of $\boldsymbol{X}$ that describes a closed, smooth surface. Define the matrices

$$\nabla_\theta \mathbf{X} = \begin{pmatrix} \frac{\partial \boldsymbol{X}}{\partial \lambda} & \frac{\partial \boldsymbol{X}}{\partial \theta} \end{pmatrix} \tag{10}$$

and

$$\nabla_\theta \mathbf{Z} = \begin{pmatrix} \frac{\partial \boldsymbol{Z}}{\partial \lambda} & \frac{\partial \boldsymbol{Z}}{\partial \theta} \end{pmatrix}, \tag{11}$$

where $\frac{\partial \boldsymbol{X}}{\partial \lambda}$ is a vector in $\mathbb{R}^3$ that represents the component-wise partial λ derivative of $\boldsymbol{X}$ (the other components of $\nabla_\theta \mathbf{X}$ and $\nabla_\theta \mathbf{Z}$ are defined similarly). The matrix $\nabla_\theta \mathbf{X}$ is the deformation gradient and the rectangular analogue of the square matrix $\mathbb{F}$ in [11]. We define the metric tensors

$$\mathcal{G} = (\nabla_\theta \mathbf{X})^{\mathrm{T}} (\nabla_\theta \mathbf{X}) = \begin{pmatrix} \frac{\partial \boldsymbol{X}}{\partial \lambda} \cdot \frac{\partial \boldsymbol{X}}{\partial \lambda} & \frac{\partial \boldsymbol{X}}{\partial \lambda} \cdot \frac{\partial \boldsymbol{X}}{\partial \theta} \\ \frac{\partial \boldsymbol{X}}{\partial \theta} \cdot \frac{\partial \boldsymbol{X}}{\partial \lambda} & \frac{\partial \boldsymbol{X}}{\partial \theta} \cdot \frac{\partial \boldsymbol{X}}{\partial \theta} \end{pmatrix} \tag{12}$$

and

$$\mathcal{G}_0 = (\nabla_\theta \mathbf{Z})^{\mathrm{T}} (\nabla_\theta \mathbf{Z}) = \begin{pmatrix} \frac{\partial \boldsymbol{Z}}{\partial \lambda} \cdot \frac{\partial \boldsymbol{Z}}{\partial \lambda} & \frac{\partial \boldsymbol{Z}}{\partial \lambda} \cdot \frac{\partial \boldsymbol{Z}}{\partial \theta} \\ \frac{\partial \boldsymbol{Z}}{\partial \theta} \cdot \frac{\partial \boldsymbol{Z}}{\partial \lambda} & \frac{\partial \boldsymbol{Z}}{\partial \theta} \cdot \frac{\partial \boldsymbol{Z}}{\partial \theta} \end{pmatrix}. \tag{13}$$

The Green–Lagrange strain energy tensor is defined as

$$\gamma = \frac{1}{2}(\mathcal{G} - \mathcal{G}_0). \tag{14}$$

In Cartesian coordinates, the Cauchy–Green deformation tensor is given by $\mathcal{C} = \mathcal{A}^{\mathrm{T}}\mathcal{A}$, where $\mathcal{A}$ is the surface displacement gradient, $\mathcal{A} = \frac{d\boldsymbol{X}}{d\boldsymbol{Z}}$ [18]. As in [12] we consider a related tensor,

$$\mathcal{C} = \mathcal{G}\mathcal{G}_0^{-1}. \tag{15}$$

In hyperelastic materials, the energy density functional W is defined in terms of two deformation invariants, I_1 and I_2, where $I_1 = \mathrm{tr}(\mathcal{C}) - 2$ and $I_2 = \det(\mathcal{C}) - 1$. As noted in [11,18], the neo-Hookean energy density can be represented as a function of the invariants, i.e. $W(I_1, I_2)$, or as a function of the deformation gradient, $\nabla_\theta \mathbf{X}$. The total energy of the surface is given by an integral of the energy density

$$\mathcal{E}(\boldsymbol{X}) = \int_{\Gamma_0} W(I_1, I_2)\, dA_0 = \int_{\Gamma_0} W(\nabla_\theta \mathbf{X})\, dA_0 = \int_{\Gamma_0} W(\nabla_\theta \mathbf{X}) \sqrt{\det \mathcal{G}_0}\, d\lambda\, d\theta, \tag{16}$$

where Γ_0 is the undeformed shell surface, and the area differential dA_0 denotes integration over the *reference* configuration. The final equality comes from the fact that $\sqrt{\det \mathcal{G}_0} = \|\boldsymbol{n_0}\|$, where $\boldsymbol{n_0} = \frac{\partial \boldsymbol{Z}}{\partial \lambda} \times \frac{\partial \boldsymbol{Z}}{\partial \theta}$ is the normal (not necessarily a unit vector) to the surface in the reference configuration, as shown in [12]. We note that the factor $\sqrt{\det \mathcal{G}_0}$ is a function of the coordinates λ and θ, but *not* of the current configuration $\boldsymbol{X}$ or its deformation gradient $\nabla_\theta \mathbf{X}$.

In order to compute the Lagrangian force density Eq. (7), consider a deformation by $\delta \boldsymbol{X}$. Evaluating Eq. (8), we have

$$\phi_{\delta \boldsymbol{X}}(\alpha) = \mathcal{E}(\boldsymbol{X} + \alpha \delta \boldsymbol{X}) = \int_{\Gamma_0} W(\nabla_\theta (\boldsymbol{X} + \alpha \delta \boldsymbol{X})) \sqrt{\det \mathcal{G}_0}\, d\lambda\, d\theta. \tag{17}$$

To compute the variational derivative, we follow the derivation from [11] and first apply the chain rule for $\nabla_\theta \mathbf{X}$ (note that the area weight $\sqrt{\det \mathcal{G}_0}$ is unaffected),

$$\left. \frac{d\phi_{\delta \boldsymbol{X}}(\alpha)}{d\alpha} \right|_{\alpha=0} = \sum_{i=1}^{3} \sum_{j=1}^{2} \int_{\Gamma_0} \frac{\partial W}{\partial (\nabla_\theta \mathbf{X})_{ij}} \frac{\partial\, \delta X_i}{\partial q_j} \sqrt{\det \mathcal{G}_0}\, d\lambda\, d\theta. \tag{18}$$

The index i describes the Cartesian coordinates (indexed from 1 to 3), and the index j describes the curvilinear coordinates λ and θ (indexed from 1 to 2). We note that the first Piola–Kirchhoff stress tensor is

$$\mathcal{P} = \frac{\partial W}{\partial(\nabla_\theta \mathbf{X})}. \tag{19}$$

Applying integration by parts on each of the coordinates λ and θ, we have

$$\left.\frac{d\phi_{\delta \boldsymbol{X}}(\alpha)}{d\alpha}\right|_{\alpha=0} = -\sum_{i=1}^{3}\sum_{j=1}^{2}\int_{\Gamma_0} \frac{\partial(\mathcal{P}_{ij}\sqrt{\det \mathcal{G}_0})}{\partial q_j}\delta X_i \, d\lambda \, d\theta. \tag{20}$$

The boundary terms evaluate to zero under the assumption that the surface is closed. We obtain an area integral by multiplying and dividing by $\sqrt{\det \mathcal{G}_0}$,

$$\left.\frac{d\phi_{\delta \boldsymbol{X}}(\alpha)}{d\alpha}\right|_{\alpha=0} = -\sum_{i=1}^{3}\sum_{j=1}^{2}\int_{\Gamma_0} \frac{1}{\sqrt{\det \mathcal{G}_0}}\frac{\partial(\mathcal{P}_{ij}\sqrt{\det \mathcal{G}_0})}{\partial q_j}\delta X_i \, dA_0. \tag{21}$$

Using the surface divergence operator in the undeformed state

$$\nabla_\theta \cdot = \frac{1}{\sqrt{\det \mathcal{G}_0}}\left(\frac{\partial}{\partial \lambda}\sqrt{\det \mathcal{G}_0}\,\boldsymbol{e_1} + \frac{\partial}{\partial \theta}\sqrt{\det \mathcal{G}_0}\,\boldsymbol{e_2}\right)\cdot, \tag{22}$$

we can write the variational derivative from Eq. (7) as

$$\frac{\delta \mathcal{E}}{\delta \boldsymbol{X}} = -\nabla_\theta \cdot \mathcal{P}^{\mathrm{T}}. \tag{23}$$

The above notation differs from [11] and instead follows [27]; it relies on a traditional tensor expansion in which $\mathcal{P}$ is written in terms of base dyads as $\sum_{i,j}\mathcal{P}_{ij}\boldsymbol{e_i}\boldsymbol{e_j}$. As shown in Appendix A, using ∇_θ from Eq. (22) and carrying out the dot product via tensor expansion in Eq. (23) yields the product in Eq. (21). The force density per unit undeformed configuration in Eq. (7) is then

$$\boldsymbol{F} = -\frac{\delta \mathcal{E}}{\delta \boldsymbol{X}} = \nabla_\theta \cdot \mathcal{P}^{\mathrm{T}}. \tag{24}$$

We point out the analogy with [11], where the surface divergence operator in Cartesian coordinates has been replaced by the curvilinear surface divergence. Although we have obtained a representation for the force, it is in practice quite cumbersome to compute $\mathcal{P} = \frac{\partial W}{\partial(\nabla_\theta \mathbf{X})}$ by directly differentiating the energy density (however, it is possible for some simple forms of W). We therefore expand the matrix $\mathcal{P}^{\mathrm{T}}$ using the chain rule with the Green–Lagrange strain tensor, γ, as an intermediate:

$$\mathcal{P}^{\mathrm{T}} = \frac{\partial W}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}} = \frac{\partial W}{\partial \gamma} : \frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}}, \tag{25}$$

where the double dot product : arises from the product of a fourth order tensor $\frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}}$ and second order tensor $\frac{\partial W}{\partial \gamma}$ to form the second order tensor $\mathcal{P}^{\mathrm{T}}$ (see Appendix A for a detailed expansion). We denote the symmetric second order tensor $\frac{\partial W}{\partial \gamma} = \mathcal{S}$, where $\mathcal{S}$ is the second Piola–Kirchhoff stress tensor [20]. We show in Appendix A that

$$\mathcal{P}^{\mathrm{T}} = \frac{\partial W}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}} = \frac{\partial W}{\partial \gamma} : \frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}} = \mathcal{S} : \frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}} = \mathcal{S} \cdot (\nabla_\theta \mathbf{X})^{\mathrm{T}}. \tag{26}$$

By the symmetry of γ (and $\mathcal{S}$), the transpose of both sides in Eq. (26) yields

$$\mathcal{P} = (\nabla_\theta \mathbf{X}) \cdot \mathcal{S}^{\mathrm{T}} = (\nabla_\theta \mathbf{X}) \cdot \mathcal{S}. \tag{27}$$

In Eq. (27), we recover the correspondence between the first and second Piola–Kirchhoff stress tensors [28].

To compute the tensor $\mathcal{S}$ in terms of the coordinates λ and θ, we begin by observing that in Eq. (15), $\mathcal{C} = \mathcal{I}_2 + 2\gamma\mathcal{G}_0^{-1}$, where $\mathcal{I}_2$ is the rank 2 identity tensor. Applying the chain rule and differentiating $\mathcal{C}$ directly with respect to γ yields

$$\mathcal{S} = \frac{\partial W}{\partial \gamma} = \frac{\partial W}{\partial \mathcal{C}} : \frac{\partial \mathcal{C}}{\partial \gamma} = 2\frac{\partial W}{\partial \mathcal{C}} \cdot \mathcal{G}_0^{-1}. \tag{28}$$

Computing $\frac{\partial W}{\partial \mathcal{C}}$ by the chain rule, we have

$$\frac{\partial W}{\partial \mathcal{C}} = \frac{\partial W}{\partial I_1}\frac{\partial I_1}{\partial \mathcal{C}} + \frac{\partial W}{\partial I_2}\frac{\partial I_2}{\partial \mathcal{C}}, \tag{29}$$

where each $\frac{\partial I_j}{\partial \mathcal{C}}$ is a second order tensor obtained from taking the derivative of each invariant I_j with respect to each element of $\mathcal{C}$. We show in Appendix A that

$$\frac{\partial I_1}{\partial \mathcal{C}} = \mathcal{I}_2 \tag{30}$$

and

$$\frac{\partial I_2}{\partial \mathcal{C}} = (\det \mathcal{C})\left(\mathcal{G}^{-1}\mathcal{G}_0\right). \tag{31}$$

Substituting Eqs. (30) and (31) into Eqs. (28) and (29),

$$\mathcal{S} = 2\frac{\partial W}{\partial \mathcal{C}}\cdot\mathcal{G}_0^{-1} = 2\left(\frac{\partial W}{\partial I_1}\frac{\partial I_1}{\partial \mathcal{C}} + \frac{\partial W}{\partial I_2}\frac{\partial I_2}{\partial \mathcal{C}}\right)\cdot\mathcal{G}_0^{-1}, \tag{32}$$

$$\mathcal{S} = 2\left(\frac{\partial W}{\partial I_1}\mathcal{I}_2 + \frac{\partial W}{\partial I_2}(\det \mathcal{C})\left(\mathcal{G}^{-1}\mathcal{G}_0\right)\right)\cdot\mathcal{G}_0^{-1}, \tag{33}$$

$$\mathcal{S} = 2\frac{\partial W}{\partial I_1}\mathcal{G}_0^{-1} + 2\frac{\partial W}{\partial I_2}(\det \mathcal{C})\,\mathcal{G}^{-1}. \tag{34}$$

Since the constitutive law is often expressed in terms of the invariants I_1 and I_2, this formulation allows us to find $\boldsymbol{F}$ with respect to λ and θ using Eq. (27).

Returning to the computation of force density as $\boldsymbol{F} = \nabla_\theta \cdot \mathcal{P}^{\mathrm{T}}$ from Eq. (24), we apply the surface divergence operator in the undeformed state in Eq. (22). Then the force density can be expressed as

$$\boldsymbol{F} = \frac{1}{\sqrt{\det \mathcal{G}_0}}\left(\frac{\partial}{\partial\lambda}\sqrt{\det \mathcal{G}_0}\left(\mathcal{S}_{11}\frac{\partial \boldsymbol{X}}{\partial\lambda} + \mathcal{S}_{12}\frac{\partial \boldsymbol{X}}{\partial\theta}\right) + \frac{\partial}{\partial\theta}\sqrt{\det \mathcal{G}_0}\left(\mathcal{S}_{21}\frac{\partial \boldsymbol{X}}{\partial\lambda} + \mathcal{S}_{22}\frac{\partial \boldsymbol{X}}{\partial\theta}\right)\right). \tag{35}$$

Further details are located in Appendix A. It is also possible to arrive at Eq. (35) by expanding the tensor $\mathcal{P}$ in Eq. (18) prior to integrating by parts.

2.4. From force density to force

The method developed in Section 2.3 allows us to find the force *density* per unit reference configuration at any point $\boldsymbol{X}$. However, because the IB method requires the *force* to be spread onto the fluid, it remains to compute the area weights associated with each point on the surface. While some applications do not require precise area weights, discretized structures in Stokes flow must have the correct area weights so that the discretized integral of force over the surface is zero. Because Eq. (35) is a translation invariant force density function, it can be shown [25] that its integral is zero in the continuous sense. The discretization of the force integral must preserve this property for structures immersed in Stokes flow. In our formulation, this condition becomes one on the surface quadrature weights. If accurate surface quadrature weights can be computed, the force density will integrate to zero discretely (within some quadrature error); the reason is that the integral of the force generated by a translation invariant hyperelastic energy over a closed surface is known to be zero [25].

Let n be the number of points at which the force is evaluated and consider the discretized integral of the force over the reference configuration

$$\int_{\Gamma_0} \boldsymbol{F}\, dA_0 \approx \sum_{i=1}^{n} \boldsymbol{F}(\boldsymbol{X^i})\,(\Delta A_0)_i. \tag{36}$$

The factors $(\Delta A_0)_i$ are the appropriate area weights. Our goal is to determine the vector of area weights $\boldsymbol{\Delta A_0} = \boldsymbol{\Delta A_0}(\boldsymbol{Z})$ (of length n) such that the integral in Eq. (36) is as close to exact as possible; that is, we are computing quadrature weights for integration over the undeformed configuration. Since the integral is over the reference configuration, the weights $\boldsymbol{\Delta A_0}$ only need to be computed once for the undeformed shape. In our applications, the undeformed shape is usually a sphere of radius 1, but the method we outline here can be used to compute the area weights in any reference configuration.

2.4.1. Weights on a sphere of radius 1

Beginning with the unit sphere $\mathbb{S}^2$, we use a method from [29] to compute nonnegative weights $\boldsymbol{\Delta A_0}$. Note that the nonnegativity constraint is necessary due to the physics of the problem; the force and force densities should point in the same directions, so the area weights should be nonnegative.

Let n be the number of points at which the force function from Section 2.3 is evaluated (we refer to these as *evaluation points*.) Following [29], we compute nonnegative quadrature weights $(\Delta A_0)_i$ such that

$$\int_{\mathbb{S}^2} f(\boldsymbol{X})\, dA_0 = \sum_{i=1}^{n} f(\boldsymbol{X^i})(\Delta A_0)_i \tag{37}$$

is exact for all functions, $f \in \Pi_N = \text{span}\left\{Y_\ell^k : \ell = 0, 1, \dots N, k = -\ell, \dots, 0, \dots \ell\right\}$. Here Π_N is the $(N+1)^2$ dimensional space of spherical harmonics up to degree N, defined as

$$Y_\ell^k(\lambda,\theta) = \begin{cases} \sqrt{\dfrac{(2\ell+1)}{4\pi}\dfrac{(\ell-k)!}{(\ell+k)!}} P_\ell^k(\sin\theta)\cos(k\lambda) & k \geq 0 \\ \sqrt{\dfrac{(2\ell+1)}{4\pi}\dfrac{(\ell+k)!}{(\ell-k)!}} P_\ell^{-k}(\sin\theta)\sin(-k\lambda) & k < 0, \end{cases} \tag{38}$$

where P_ℓ^k is an associated Legendre function of degree ℓ and order k. Let $f(\boldsymbol{X}) = f(\lambda,\theta) = Y_0^0(\lambda,\theta)Y_\ell^k(\lambda,\theta)$. Then to satisfy Eq. (37),

$$\int_{\mathbb{S}^2} f(\boldsymbol{X})\, d\boldsymbol{x} = \iint_{\mathbb{S}^2} Y_0^0(\lambda,\theta)Y_\ell^k(\lambda,\theta)\, dA_0 \tag{39}$$

$$= \sum_{i=1}^{n} Y_0^0(\lambda_i,\theta_i)Y_\ell^k(\lambda_i,\theta_i)(\Delta A_0)_i \tag{40}$$

$$= \frac{1}{\sqrt{4\pi}} \sum_{i=1}^{n} Y_\ell^k(\lambda_i,\theta_i)(\Delta A_0)_i. \tag{41}$$

The last equality is due to the fact that $Y_0^0 = \dfrac{1}{\sqrt{4\pi}}$. Since spherical harmonics form an orthonormal set,

$$\int_{\mathbb{S}^2} f(\boldsymbol{X})\, d\boldsymbol{x} = \iint_{\mathbb{S}^2} Y_0^0(\lambda,\theta)Y_\ell^k(\lambda,\theta)\, dA_0 = \begin{cases} 1 & \ell = 0 \\ 0 & \ell \neq 0. \end{cases} \tag{42}$$

Equating Eqs. (41) and (42), we have that

$$\sum_{i=1}^{n} Y_\ell^k(\lambda_i,\theta_i)(\Delta A_0)_i = \begin{cases} \sqrt{4\pi} & \ell = 0 \\ 0 & \ell \neq 0. \end{cases} \tag{43}$$

Let the matrix

$$\mathbf{Y} = \begin{pmatrix} Y_0^0(\lambda_1,\theta_1) & Y_1^0(\lambda_1,\theta_1) & Y_1^{-1}(\lambda_1,\theta_1) & Y_1^1(\lambda_1,\theta_1) & \dots \\ Y_0^0(\lambda_2,\theta_2) & Y_1^0(\lambda_2,\theta_2) & Y_1^{-1}(\lambda_2,\theta_2) & Y_1^1(\lambda_2,\theta_2) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Y_0^0(\lambda_n,\theta_n) & Y_1^0(\lambda_n,\theta_n) & Y_1^{-1}(\lambda_n,\theta_n) & Y_1^1(\lambda_n,\theta_n) & \dots \end{pmatrix} \in \mathbb{C}^{n\times(N+1)^2}. \tag{44}$$

Then in Eq. (43) we seek the nonnegative least squares solution to

$$\mathbf{Y}^*\boldsymbol{\Delta A_0} = \sqrt{4\pi}\boldsymbol{e_1}, \tag{45}$$

where $\boldsymbol{e_1} = (1 \quad 0 \quad \dots \quad 0)^T$ is an $(N+1)^2$ dimensional vector. The nonnegative least squares solution for $\boldsymbol{\Delta A_0}$ gives the area weights for Eq. (37).

To solve Eq. (45) for $\boldsymbol{\Delta A_0}$, we use the available implementation from [30] to obtain the nonnegative area weights $\boldsymbol{\Delta A_0}$ for a unit sphere. Although any point set can be used in Eq. (45), we let $n = (N+1)^2$ and choose evaluation points that maximize the determinant of the Gram matrix $\mathbf{YY}^*$. For these "maximal determinant" (MD) points, which are discussed in

[31,32] and can be downloaded from [33], it has been conjectured that the quadrature weights satisfying the equality in Eq. (45) exist and are positive for all N [32].

Choosing weights that satisfy Eq. (45) ensures that the integral in Eq. (37) is exact for functions in Π_N, the space of spherical harmonics up to degree N. However, the calculated weights can still be used to integrate any function on the sphere. In fact, the force function in Eq. (35) is not necessarily in Π_N since it involves derivatives of arbitrary position functions. A quadrature error is therefore introduced. In [32], it is shown that the quadrature error, ϵ, in a certain Hilbert space is bounded by

$$\epsilon \leq \frac{\sqrt{4\pi}}{(N+1)^{1/2}} = \frac{\sqrt{4\pi}}{n^{1/4}}, \tag{46}$$

where the last equality holds only in the case that $n = (N+1)^2$. For MD points, the measured worst case error is much better; it is $\mathcal{O}(n^{-3/4})$ (see [32], Table 2). Our observations in Section 4.1 confirm this, as the actual quadrature error in integrating the force function was found to be several orders of magnitude lower than the theoretical bound, even for a highly irregular shape.

2.4.2. Weights on an arbitrary reference configuration

We use the formulation from [34] to determine the quadrature weights on an arbitrary configuration. Let $\mathbb{L}$ denote a space diffeomorphic to $\mathbb{S}^2$, and let $\boldsymbol{Z}_{\mathbb{S}^2}(\lambda,\theta)$ and $\boldsymbol{Z}_{\mathbb{L}}(\lambda,\theta)$ be the mappings from Eq. (9) that describe $(\hat{X},\hat{Y},\hat{Z})$ from (λ,θ) on each space. Then, as shown in [34], the surface elements for each configuration are given by

$$dA_{\mathbb{L}} = \sqrt{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{L}}\right)\right)}\, d\lambda\, d\theta \tag{47}$$

and

$$dA_{\mathbb{S}^2} = \sqrt{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{S}^2}\right)\right)}\, d\lambda\, d\theta, \tag{48}$$

where $\mathcal{G}_0$ is evaluated for each surface according to Eq. (13). From Eqs. (47) and (48), it is clear that

$$dA_{\mathbb{L}} = \sqrt{\frac{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{L}}\right)\right)}{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{S}^2}\right)\right)}}\, dA_{\mathbb{S}^2}. \tag{49}$$

Thus the analogous integral to Eq. (36) is

$$\int_{\mathbb{L}} f(\boldsymbol{X})\, dA_{\mathbb{L}} = \int_{\mathbb{S}^2} f(\boldsymbol{X}) \sqrt{\frac{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{L}}\right)\right)}{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{S}^2}\right)\right)}}\, dA_{\mathbb{S}^2} = \sum_{i=1}^{n} f(\boldsymbol{X}^{\boldsymbol{i}}) \sqrt{\frac{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{L}}\right)\right)}{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{S}^2}\right)\right)}}\, (\Delta A_0)_i\,. \tag{50}$$

According to Eq. (50), the weights for the arbitrary reference configuration are given by

$$\boldsymbol{\Delta A}_{\mathbb{L}} = \sqrt{\frac{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{L}}\right)\right)}{\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{S}^2}\right)\right)}}\, \boldsymbol{\Delta A}_{\mathbb{S}^2}. \tag{51}$$

For example, if the reference configuration is a sphere of radius r, then $\det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{L}}\right)\right) = r^4 \det\left(\mathcal{G}_0\left(\boldsymbol{Z}_{\mathbb{S}^2}\right)\right)$, so $\boldsymbol{\Delta A}_{\mathbb{L}} = r^2 \boldsymbol{\Delta A}_{\mathbb{S}^2}$, as expected since the surface area of a sphere scales by r^2.

With the area weights from Eq. (51), we have finally found the *force*, $\widehat{\boldsymbol{F}}$ at point i from the force density $\boldsymbol{F}$ as a function of position in (λ,θ),

$$\widehat{\boldsymbol{F}}_i(\lambda,\theta) = \boldsymbol{F}_i(\lambda,\theta)(\Delta A_0)_i. \tag{52}$$

2.5. The linear discrete surface method (LDSM)

In LDSM (first introduced in [26]), the surface of the shell is triangulated and the integral for energy is discretized as such. Consider an energy functional from Eq. (6) of the form

$$\mathcal{E} = \int_{\Gamma_0} W(I_1, I_2)\, dA_0. \tag{53}$$

The above integral can be discretized into a sum over the surface triangles as follows,

$$\mathcal{E} = \sum_{\text{tri}} \int_{\text{tri}} W(I_1, I_2)\, dA_0. \tag{54}$$

LDSM uses barycentric coordinates $(\gamma_1, \gamma_2, \gamma_3)$ to represent a deformed triangle t with vertices $\boldsymbol{X}^1, \boldsymbol{X}^2, \boldsymbol{X}^3$ in the deformed configuration that corresponds to a reference triangle s with vertices $\boldsymbol{Z}^1, \boldsymbol{Z}^2, \boldsymbol{Z}^3$ in the undeformed configuration. Any point within the triangle can be written as

$$\boldsymbol{X} = \gamma_1 \boldsymbol{X}^1 + \gamma_2 \boldsymbol{X}^2 + \gamma_3 \boldsymbol{X}^3 \tag{55}$$

and

$$\boldsymbol{Z} = \gamma_1 \boldsymbol{Z}^1 + \gamma_2 \boldsymbol{Z}^2 + \gamma_3 \boldsymbol{Z}^3, \tag{56}$$

where $\gamma_1 + \gamma_2 + \gamma_3 = 1$ and $\gamma_1, \gamma_2, \gamma_3 \geq 0$. Using the relation $\gamma_3 = 1 - \gamma_1 - \gamma_2$, the surface can be represented in terms of two material coordinates (note that, for $\gamma_3 \geq 0$ to be satisfied, $\gamma_1 \leq 1 - \gamma_2$). As in the continuous case, we define the matrices

$$\nabla_{\mathbf{q}} \mathbf{X} = \begin{pmatrix} \dfrac{\partial \boldsymbol{X}}{\partial \gamma_1} & \dfrac{\partial \boldsymbol{X}}{\partial \gamma_2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^1 - \boldsymbol{X}^3 & \boldsymbol{X}^2 - \boldsymbol{X}^3 \end{pmatrix} \tag{57}$$

and

$$\nabla_{\mathbf{q}} \mathbf{Z} = \begin{pmatrix} \dfrac{\partial \boldsymbol{Z}}{\partial \gamma_1} & \dfrac{\partial \boldsymbol{Z}}{\partial \gamma_2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{Z}^1 - \boldsymbol{Z}^3 & \boldsymbol{Z}^2 - \boldsymbol{Z}^3 \end{pmatrix}. \tag{58}$$

Denote $\boldsymbol{X}^i - \boldsymbol{X}^j = \boldsymbol{X}^{ij}$ (and likewise for $\boldsymbol{Z}$). Then we can again define the metric tensors

$$\mathcal{G} = (\nabla_{\mathbf{q}} \mathbf{X})^{\mathrm{T}} (\nabla_{\mathbf{q}} \mathbf{X}) = \begin{pmatrix} \boldsymbol{X}^{13} \cdot \boldsymbol{X}^{13} & \boldsymbol{X}^{13} \cdot \boldsymbol{X}^{23} \\ \boldsymbol{X}^{23} \cdot \boldsymbol{X}^{13} & \boldsymbol{X}^{23} \cdot \boldsymbol{X}^{23} \end{pmatrix} \tag{59}$$

and

$$\mathcal{G}_0 = (\nabla_{\mathbf{q}} \mathbf{Z})^{\mathrm{T}} (\nabla_{\mathbf{q}} \mathbf{Z}) = \begin{pmatrix} \boldsymbol{Z}^{13} \cdot \boldsymbol{Z}^{13} & \boldsymbol{Z}^{13} \cdot \boldsymbol{Z}^{23} \\ \boldsymbol{Z}^{23} \cdot \boldsymbol{Z}^{13} & \boldsymbol{Z}^{23} \cdot \boldsymbol{Z}^{23} \end{pmatrix}. \tag{60}$$

The tensors $\mathcal{G}$ and $\mathcal{G}_0$ are *constant* on any triangle t or s. Therefore, the tensor $\mathcal{C} = \mathcal{G}\mathcal{G}_0^{-1}$ and its invariants I_1 and I_2 are constant on t and s. Thus any energy density functional defined in terms of the invariants is also constant on t and s and can be defined in terms of the vertices $\boldsymbol{X}^1, \boldsymbol{X}^2, \boldsymbol{X}^3, \boldsymbol{Z}^1, \boldsymbol{Z}^2,$ and $\boldsymbol{Z}^3$. Denote the energy density of a triangle as $W(t, s)$ (it is a function of the deformed triangle t and undeformed triangle s). The discretized energy is then

$$\mathcal{E} = \sum_{\text{tri}} \int_s W(I_1, I_2)\, dA_0 = \sum_{\text{tri}} \int_s W(t, s) (\det \mathcal{G}_0)^{1/2}\, d\gamma_1\, d\gamma_2. \tag{61}$$

Note that the integration is carried out over the *reference* triangle s and that $dA_0 = (\det \mathcal{G}_0)^{1/2}\, d\gamma_1\, d\gamma_2$ since $(\det \mathcal{G}_0)^{1/2}$ is the magnitude of the normal to the reference triangle [12]. Evaluating the integral, we have

$$\mathcal{E} = \sum_{\text{tri}} W(t, s) (\det \mathcal{G}_0)^{1/2} \int_s d\gamma_2\, d\gamma_1, \tag{62}$$

$$\mathcal{E} = \sum_{\text{tri}} W(t, s) (\det \mathcal{G}_0)^{1/2} \int_0^1 \int_0^{1-\gamma_1} d\gamma_2\, d\gamma_1, \tag{63}$$

$$\mathcal{E} = \frac{1}{2} \sum_{\text{tri}} W(t, s) (\det \mathcal{G}_0)^{1/2}. \tag{64}$$

From the above equation, note that $\Delta A_0(s) = \dfrac{1}{2} (\det \mathcal{G}_0)^{1/2}$, which is expected *a priori* since it is simply the formula for the area of the reference triangle s. Since the energy functional has already been integrated, the area weights at each point have already been accounted for, and the *force* (not force density), $\widehat{\boldsymbol{F}}$, at a point $\boldsymbol{X}^i$ due to triangle t can be computed by taking the derivative of the energy with respect to that point,

$$\widehat{\boldsymbol{F}}_i(t) = -\frac{\partial \mathcal{E}}{\partial \boldsymbol{X}^i}(t) = -\frac{\partial}{\partial \boldsymbol{X}^i} \left(\frac{1}{2} W(t, s) (\det \mathcal{G}_0)^{1/2} \right). \tag{65}$$

The force at each point is then the sum of the force contributions from each triangle that includes the point, i.e.

$$\widehat{\boldsymbol{F}}_i\left(\boldsymbol{X}^i\right)=\sum_{t\ni\boldsymbol{X}^i}-\frac{\partial\mathcal{E}}{\partial\boldsymbol{X}^i}(t)=\sum_{t\ni\boldsymbol{X}^i}-\frac{\partial}{\partial\boldsymbol{X}^i}\left(\frac{1}{2}W(t,s)(\det\mathcal{G}_0)^{1/2}\right). \tag{66}$$

It can be shown using vector calculus that the sum of the forces over the points in LDSM is zero. The force in Eq. (66) therefore satisfies the discrete integral constraint for Stokes flow. The force density per unit undeformed area is computed by dividing the force at each point by its associated area weight in the undeformed configuration. The area weight at point i in LDSM is given by 1/3 of the sum of the reference triangle areas that include point i. That is,

$$\Delta(A_{L_0})_i=\frac{1}{3}\sum_{s\ni\boldsymbol{Z}^i}(\det\mathcal{G}_0(s))^{1/2}. \tag{67}$$

Further details are located in [12].

2.6. Energy functionals

In this section, we describe energy functionals for neo-Hookean energy, surface tension, and bending (or curvature) energy.

2.6.1. Neo-Hookean energy

We follow Evans & Skalak's formulation for the neo-Hookean energy density [35] (for other possible neo-Hookean constitutive laws, see [18]). In terms of the invariants,

$$W_{NH}=G_s\left(\frac{I_1+2}{2\sqrt{I_2+1}}-1+\frac{A}{2}\left(I_2+2-2\sqrt{I_2+1}\right)\right) \tag{68}$$

$$=G_s\left(\frac{\operatorname{tr}\mathcal{C}}{2(\det\mathcal{C})^{1/2}}-1+\frac{A}{2}\left(\det\mathcal{C}+1-2(\det\mathcal{C})^{1/2}\right)\right) \tag{69}$$

$$=G_s\left(\frac{\operatorname{tr}\mathcal{C}}{2(\det\mathcal{C})^{1/2}}-1+\frac{A}{2}\left((\det\mathcal{C})^{1/2}-1\right)^2\right) \tag{70}$$

$$=\frac{K}{2}\left((\det\mathcal{C})^{1/2}-1\right)^2+\frac{G_s}{2}\left(\frac{\operatorname{tr}\mathcal{C}}{(\det\mathcal{C})^{1/2}}-2\right), \tag{71}$$

where $K=AG_s$ is the area dilation modulus and G_s is the shear modulus. Eq. (71) gives the form used in [12]. We take derivatives of Eq. (68) with respect to I_1 and I_2 to find the tensors $\mathcal{P}$, $\mathcal{S}$, and force density $\boldsymbol{F}$ in Eq. (35).

2.6.2. Surface energy

The most common representation of "surface energy" is $\mathcal{E}_{ST}=\sigma\int dA$, where dA is the area element in the *current* configuration and σ is the surface tension (with units of force/length) of the surface. For both the variational force density method and LDSM, $dA=(\det\mathcal{G})^{1/2}\,d\lambda\,d\theta$, but this must be expressed with respect to the integral over the reference area to satisfy our method. Therefore we use Eq. (50) and [34] to write

$$\mathcal{E}_{ST}=\int_\Gamma\sigma\,dA=\int_{\Gamma_0}\sigma\left(\frac{\det\mathcal{G}}{\det\mathcal{G}_0}\right)^{1/2}dA_0=\int_{\Gamma_0}\sigma\,(\det\mathcal{C})^{1/2}\,dA_0. \tag{72}$$

We can then express the surface energy in the form of Eq. (16) with $W_{ST}=\sigma\,(\det\mathcal{C})^{1/2}=\sigma\,(I_2+1)^{1/2}$. Derivatives of this density with respect to I_1 and I_2 can easily be calculated to evaluate the first and second Piola–Kirchhoff stress tensors $\mathcal{P}$, $\mathcal{S}$ and force density $\boldsymbol{F}$ in Eq. (35).

The variational derivative can be carried out over the current configuration as well to yield force density per unit deformed area [15,36,37]. When this formulation, which is related directly to the mean curvature of the surface, is used, a transformation must be made to express the force per unit reference area.

$$\boldsymbol{F}^{ST}_{cur}\,dA=\sigma(2H)\,\hat{\boldsymbol{n}}\,dA=\sigma(2H)\,\hat{\boldsymbol{n}}\left(\frac{\det\mathcal{G}}{\det\mathcal{G}_0}\right)^{1/2}dA_0. \tag{73}$$

The force per unit undeformed configuration then is,

$$\boldsymbol{F}^{ST}=\sigma(2H)\,\hat{\boldsymbol{n}}\left(\frac{\det\mathcal{G}}{\det\mathcal{G}_0}\right)^{1/2}, \tag{74}$$

where $\hat{\boldsymbol{n}}$ is the unit normal vector in the current configuration,

$$\hat{\boldsymbol{n}} = \frac{\frac{\partial \boldsymbol{X}}{\partial \lambda} \times \frac{\partial \boldsymbol{X}}{\partial \theta}}{\left\| \frac{\partial \boldsymbol{X}}{\partial \lambda} \times \frac{\partial \boldsymbol{X}}{\partial \theta} \right\|}. \tag{75}$$

H is the mean curvature (one half the total curvature, or the average of the two principle curvatures) given by the formula

$$H = \frac{eG - 2fF + gE}{2(EG - F^2)}, \tag{76}$$

where E, F, and G are coefficients of the first fundamental form

$$E = \frac{\partial \boldsymbol{X}}{\partial \lambda} \cdot \frac{\partial \boldsymbol{X}}{\partial \lambda} \qquad F = \frac{\partial \boldsymbol{X}}{\partial \lambda} \cdot \frac{\partial \boldsymbol{X}}{\partial \theta} \qquad G = \frac{\partial \boldsymbol{X}}{\partial \theta} \cdot \frac{\partial \boldsymbol{X}}{\partial \theta}, \tag{77}$$

and e, f, and g are coefficients of the second fundamental form,

$$e = \frac{\partial^2 \boldsymbol{X}}{\partial \lambda^2} \cdot \hat{\boldsymbol{n}} \qquad f = \frac{\partial^2 \boldsymbol{X}}{\partial \lambda \partial \theta} \cdot \hat{\boldsymbol{n}} \qquad g = \frac{\partial^2 \boldsymbol{X}}{\partial \theta^2} \cdot \hat{\boldsymbol{n}}. \tag{78}$$

It can be shown using elementary vector calculus that Eq. (74) is equivalent to Eq. (35) when $W_{ST} = \sigma(I_2 + 1)^{1/2}$ is used. We emphasize that for this simple case, we were able to take a surface energy given over the current configuration and transform it so we could integrate over the reference configuration and apply our method.

More complex energy functionals given over the current configuration are not as easily transformed. In these cases, differential geometry can be used to find the variational derivatives and force per unit deformed area. These expressions can then be transformed to the undeformed configuration by multiplying by the ratio of area weights as in Eq. (74). Bending energy is an example of such an energy functional and is described in the next section.

2.6.3. Bending energy

In this section, we describe the variational formulation of forces due to bending. It is worth noting that piecewise linear representations of the surface (such as LDSM) must use a method different from that of Section 2.5, where the energy is evaluated on each triangle, to compute forces due to bending. This is because bending energies are directly related to curvature, and in a piecewise linear surface representation the triangles have zero curvature. In the case of [12], the method used to compute forces due to bending does not ensure that the bending force integrates to zero (despite the bending energy being translation invariant), and therefore cannot be used in the context of Stokes flow. Our method allows for the true bending energy functional to be used because we can properly resolve the curvature of the surface. In addition, the use of the correct quadrature weights ensures that the bending force integrates to zero over the fluid domain.

Interestingly, variational principles have been used to derive forces due to bending rigidity for quite some time, with one early example from [38]. These previous studies addressed the problem of determining the equilibrium shape of membranes experiencing bending forces by setting the force density equal to zero. The derivation of the force density due to bending begins with the energy functional

$$\mathcal{E}_{bend} = k_{bend} \int_{\Gamma} (2H)^2 \, dA, \tag{79}$$

where k_{bend} is the bending rigidity and the energy is an integral of the total curvature ($2H$) over the *deformed* configuration. The variational approach calls for the energy functional to be written in the form

$$\delta \mathcal{E}_{bend} = \int_{\Gamma} -\boldsymbol{F}^{BR}_{cur} \cdot \delta \boldsymbol{X} \, dA. \tag{80}$$

As shown in [36,39], arguments from differential geometry can be used to write the force density per unit deformed configuration as

$$\boldsymbol{F}^{BR}_{cur} = -k_{bend} \left(4\nabla_s^2 H + 8H^3 - 4HR \right) \hat{\boldsymbol{n}}, \tag{81}$$

where H is the mean curvature defined according to Eq. (76), $\hat{\boldsymbol{n}}$ is the unit normal defined in Eq. (75), and ∇_s^2 is the surface Laplacian in the deformed state, defined as

$$\nabla_s^2 = \sum_{i=1}^{2} \sum_{j=1}^{2} \frac{1}{\sqrt{\det \mathcal{G}}} \frac{\partial}{\partial q_i} \left(\sqrt{\det \mathcal{G}} \, \mathcal{G}^{-1}_{ij} \frac{\partial}{\partial q_j} \right). \tag{82}$$

Finally, R is the scalar curvature, or twice the Gaussian curvature, defined as

$$R = 2\det\left(\mathcal{G}^{-1}\begin{pmatrix} e & f \\ f & g\end{pmatrix}\right), \tag{83}$$

where the fundamental forms e, f, and g are defined in Eq. (78). In a similar manner to that of Eq. (74), Eq. (81) can be written per unit undeformed area as

$$\boldsymbol{F}^{BR} = -k_{bend}\left(\frac{\det\mathcal{G}}{\det\mathcal{G}_0}\right)^{1/2}\left(4\nabla_s^2 H + 8H^3 - 4HR\right)\hat{\boldsymbol{n}}, \tag{84}$$

which gives the force due to bending per unit undeformed area.

3. Numerical formulation

In this section, we describe the discretization of the surface and fluid grids. We then describe the fluid solver and temporal update for dynamic simulations.

3.1. Lagrangian surface and force discretization

For the variational force density method, some mapping must be chosen for $\boldsymbol{X}$ in Eq. (9). Our method allows any mapping that defines a closed surface to be used, including interpolants composed of Lagrange, spherical harmonic, or radial basis functions. We choose to use a spherical harmonic interpolant for the maps X, Y, and Z in Eq. (9) (the reference configuration $\boldsymbol{Z}$ is generally taken to be the unit sphere unless otherwise specified). Each interpolant is computed in the same manner and is also described in [15]. Suppose we have a set of $m = (M+1)^2$ points in (x, y, z) space. Then we use a spherical harmonic interpolant of order M to approximate the function $\boldsymbol{X}(\lambda,\theta)$ in Eq. (9). For example,

$$X(\lambda,\theta) = \sum_{\ell=0}^{M}\left(\sum_{k=-\ell}^{-1} c^x_{\ell,k}Y^k_\ell(\lambda,\theta) + \sum_{k=0}^{\ell} c^x_{\ell,k}Y^k_\ell(\lambda,\theta)\right), \tag{85}$$

where the spherical harmonics are as defined in Eq. (38) and the other interpolants Y, Z, $\hat{X}$, $\hat{Y}$, and $\hat{Z}$ are defined similarly. In [15], it is shown that Eq. (85) corresponds to a linear system

$$\mathbf{Y}\mathbf{c}^{\boldsymbol{x}} = \begin{pmatrix} Y_0^0(\lambda_1,\theta_1) & Y_1^0(\lambda_1,\theta_1) & Y_1^{-1}(\lambda_1,\theta_1) & Y_1^1(\lambda_1,\theta_1) & \dots \\ Y_0^0(\lambda_2,\theta_2) & Y_1^0(\lambda_2,\theta_2) & Y_1^{-1}(\lambda_2,\theta_2) & Y_1^1(\lambda_2,\theta_2) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Y_0^0(\lambda_m,\theta_m) & Y_1^0(\lambda_m,\theta_m) & Y_1^{-1}(\lambda_m,\theta_m) & Y_1^1(\lambda_m,\theta_m) & \dots \end{pmatrix}\begin{pmatrix} c_1^x \\ c_2^x \\ \vdots \\ c_m^x \end{pmatrix} = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{pmatrix}, \tag{86}$$

where $m = (M+1)^2$ exactly and $X_1, X_2, \dots, X_m$ are the x coordinates of the first, second, and mth points, respectively. Since the coordinates of the interpolation points are constant throughout the algorithm, the LU factorization of the matrix $\mathbf{Y}$ in Eq. (86) can be precomputed at the start of the algorithm. Then at each timestep, $\mathcal{O}(m^2)$ flops are required to solve Eq. (86).

For the variational force density method, we follow [15] and define the m points in (x, y, z) space that are used to construct the interpolant as *interpolation* points. Note that there are typically much fewer interpolation points than evaluation points ($m \ll n$). Because the matrix in Eq. (86) must be invertible, we use the maximal determinant points mentioned previously (downloaded from [33]) that maximize the determinant of the Gram matrix $\mathbf{Y}\mathbf{Y}^*$. See [31] for more details on choosing points on a sphere for interpolation.

The interpolant from Eq. (85) is the map $\boldsymbol{X}(\lambda,\theta)$ in Eq. (9), and given a reference configuration $\boldsymbol{Z}(\lambda,\theta)$ we proceed in computing force densities from the variational method in Eq. (35) at a different set of n maximal determinant evaluation points. We find the spherical coordinates of the evaluation points and use the function $\boldsymbol{X}$ to map them to our deformed configuration, where we evaluate the forces using Eq. (35) with the area weights from Eq. (45). The calculation of the forces in Eq. (35) requires the derivatives of $\boldsymbol{X}$ at each of the evaluation points. In practice, this is accomplished by multiplying matrices of spherical harmonic derivatives (e.g. $\dfrac{\partial\mathbf{Y}}{\partial\lambda}$, where the derivatives are taken on each element) by the vector of function weights ($\boldsymbol{c^j}$). Each derivative is thus a matrix-vector multiplication problem and requires $\mathcal{O}(mn)$ operations, although this could easily be parallelized to reduce computational time. Once the derivatives are computed, $\mathcal{O}(n)$ operations are required to compute the forces in Eq. (35) from the derivative values (this step could also easily be done in parallel). The leading order term in the flop count for the force calculation in our method is therefore $\mathcal{O}(mn)$.

In LDSM, the force is computed directly at each of the evaluation points using Eq. (66) with the triangles from the surface discretization. Since the force is computed triangle by triangle and there are approximately $2n$ triangles, the entire

force calculation for LDSM is $\mathcal{O}(n)$. We perform timing tests in Section 4.2 to compare the computational costs of our method with LDSM in practice.

Each of the weights $(\Delta A_0)_i$ in Eq. (45) is computed for the n evaluation points and multiplied by its respective force density to get a total force at each of the evaluation points. Since the continuous force density representation is exact for a given position function $\boldsymbol{X}$, the only errors introduced are in representing the surface by a continuously differentiable spherical harmonic interpolant and in using a quadrature rule to integrate the continuous force density function. A detailed study of the errors associated with a spherical harmonic representation can be found in [15]. An analytic estimate for the quadrature error is located in [32]. We also analyze the error in force computations and compare it to the error in LDSM in Section 4.1.

3.2. Eulerian discretization

The fluid domain is discretized on a periodic three dimensional domain $[-L, L] \times [-L, L] \times [-L, L]$ with mesh spacing $\Delta x = \Delta y = \Delta z = 2L/\eta$, where η is the grid size. We use a spectral fluid solver to compute the pressure and velocity at each time step in $\mathcal{O}(\eta^3 \log \eta^3)$ operations [40]. The evaluation points are chosen so that approximately two Lagrangian points lie within an Eulerian cube of dimension Δx^3.

3.3. Time update

We outline the time stepping procedure for the variational derivative method first. Given a set of m interpolation points and n evaluation points on the reference configuration, and the map $\boldsymbol{Z}(\lambda, \theta)$ in Eq. (9), the area weights $\boldsymbol{\Delta A_0}$ in Eq. (45), and matrix $\mathbf{Y}$ in Eq. (86) (and its LU factorization) are precomputed. Then at the kth time step,

1. The m interpolation points in the current configuration (x_i, y_i, z_i) are used to solve Eq. (86) for the map $\boldsymbol{X}$ ($\mathcal{O}(m^2)$ operations).
2. The computed maps $\boldsymbol{X}$ and $\boldsymbol{Z}$ are used to find the force densities in Eq. (35) at each of the n *evaluation* points ($\mathcal{O}(mn)$ operations).
3. The forces are spread onto the Eulerian grid using the discrete delta function from [25] with the area weights from Eq. (45) that ensure the spread force integrates to zero ($\mathcal{O}(4^3 n)$ operations because we are using a 4 point discrete delta function).
4. The Stokes equations are solved on the Eulerian grid ($\mathcal{O}(\eta^3 \log \eta^3)$ operations).
5. The fluid velocity is interpolated back to the structure at the m *interpolation* points ($\mathcal{O}(4^3 m)$ operations).
6. The interpolation points are updated with the local fluid velocity,

$$\boldsymbol{X}_i^{k+1} = \boldsymbol{X}_i^k + \Delta t\, \boldsymbol{U}^{k+1}, \tag{87}$$

where $\boldsymbol{U}^{k+1}$ indicates the interpolated fluid velocity from step 5 ($\mathcal{O}(m)$ operations).

In LDSM, we work with only the set of n evaluation points. We begin with a triangulated surface and use the locations of the points in the current and reference configurations to compute the forces, then proceed with steps 3–6 as above, with the exception being that the structure velocity is computed at the same n points where the force was evaluated. The interpolation of the velocity in LDSM therefore requires $\mathcal{O}(4^3 n)$ operations instead of $\mathcal{O}(4^3 m)$, and the forward Euler update is $\mathcal{O}(n)$ in LDSM instead of $\mathcal{O}(m)$ in our method. If $m \ll n$, our method results in fewer operations in steps 5 and 6.

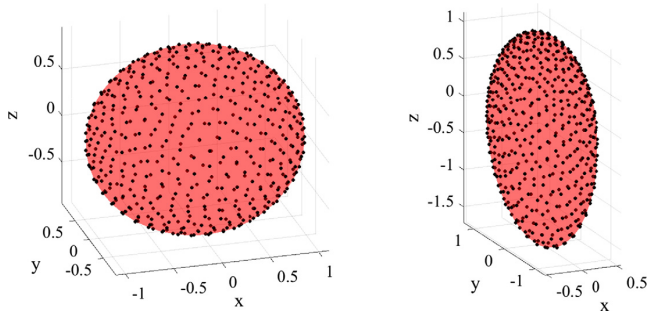
4. Results

In this section we test the accuracy of LDSM and the spherical harmonic/variational derivative (SHVD) approaches in the context of both force computations and dynamic immersed boundary simulations. We first consider the accuracy of the force computation on an ellipsoid and perturbed ellipsoid and proceed with dynamic simulations on each surface.

4.1. Accuracy of force computations

We consider the accuracy of surface tension and neo-Hookean forces on two surfaces given by

$$\text{Ellipsoid: } \boldsymbol{X} = \begin{pmatrix} a\cos\lambda\cos\theta \\ b\sin\lambda\cos\theta \\ c\sin\theta \end{pmatrix} \tag{88}$$



(a) Ellipsoid (b) Perturbed ellipsoid

Fig. 1. The shapes for study. 529 evaluation points are shown. The distribution of other sets of evaluation points is similar.

and

$$\text{“Perturbed ellipsoid”: } \boldsymbol{X} = V_f \begin{pmatrix} a\left(1+\frac{B}{5}\exp(-\sin\theta)\right)\cos\lambda\cos\theta \\ b\,(1+B\exp(-\sin\theta))\sin\lambda\cos\theta \\ c\,(1+B\exp(-\sin\theta))\sin\theta \end{pmatrix}. \tag{89}$$

We use parameters $a = 1.1$, and $b = c = \frac{1}{\sqrt{1.1}}$ in Eq. (88) and $a = 0.1$, $b = c = 0.2$, and $B = 0.25$ in Eq. (89). In both cases, the reference configuration is the unit sphere, $\boldsymbol{Z} = (\cos\lambda\cos\theta, \sin\lambda\cos\theta, \sin\theta)$, and the parameter V_f in Eq. (89) is set so that the volume of the perturbed ellipsoid is the same as that of the unit sphere (in our case, $V_f \approx 5.14$). For simplicity, we also let $\sigma = K = A = G_s = 1$ in the force and energy functionals in Eqs. (68) and (74). The shapes are shown in Fig. 1 along with the positions of the set of 529 evaluation points (sets with a higher number of evaluation points are similarly distributed).

We fix a number of evaluation points so that the quadrature error given by the weights in Eq. (45) is on the order 10^{-5} or less. The quadrature error is measured by the value of the discrete force integral since the exact integral of the translation invariant continuous force function evaluates to zero. We then compute the exact solution for the neo-Hookean or surface tension force densities at the n evaluation points using Eq. (35) to take the variational derivative of *the exact* mappings $\boldsymbol{X}$ from Eqs. (88) or (89).

Next, we construct a spherical harmonic interpolant for the surface of the form in Eq. (85) using some number m of interpolation points. We then use this interpolant to compute the force densities in Eq. (35). We multiply both the exact force density and the force density from SHVD by the weights in Eq. (45) to get a force. In this way, we isolate the error from representing the surface with a spherical harmonic interpolant. Finally, we compute the LDSM forces by evaluating Eq. (66) with the triangle vertex locations given by Eq. (88) or (89).

We denote the exact force (not force density) as $\widehat{\boldsymbol{F}}_E$, the force from SHVD as $\widehat{\boldsymbol{F}}_S$ and the force from LDSM as $\widehat{\boldsymbol{F}}_L$. We quantify the error between the models via the discrete ℓ_∞ norm. The norm of the errors from SHVD and LDSM, denoted by $\boldsymbol{E_S}$ and $\boldsymbol{E_L}$, respectively, are given by

$$\|\boldsymbol{E_S}\|_\infty = \max_i \left\| \widehat{\boldsymbol{F}}_S - \widehat{\boldsymbol{F}}_E \right\|_\infty \tag{90}$$

and

$$\|\boldsymbol{E_L}\|_\infty = \max_i \left\| \widehat{\boldsymbol{F}}_L - \widehat{\boldsymbol{F}}_E \right\|_\infty. \tag{91}$$

The error in force is computed as the maximum over the n evaluation points, where i is the index that runs from 1 to n. In all convergence plots, we plot the error vs. $\sqrt{m}$ or $\sqrt{n}$. Since $(dA_0)_i \approx 4\pi/n$, the number of points is proportional to the square of the length scale. We plot convergence in the length scale by taking the square root of the number of points.

Because the same area weights are used for the calculation of $\widehat{\boldsymbol{F}}_E$ and $\widehat{\boldsymbol{F}}_S$, the measured error in Eq. (90) is independent of the quadrature weights. Since LDSM uses different area weights, the measured error in Eq. (91) is affected by both the quadrature error in $\widehat{\boldsymbol{F}}_E$ and the discretization error in the LDSM force calculation. We define $\widehat{\boldsymbol{F}}_E$ to be the exact solution if the quadrature error (i.e. the distance of the force integral from zero) is negligible relative to the LDSM surface discretization error. We only consider surfaces where the quadrature error is $\mathcal{O}(10^{-5})$ or less. In our numerical tests, the LDSM error graphed in Figs. 2 and 3(a) is larger than this value so that quadrature error is negligible.

Fig. 2 shows the errors in Eqs. (90) and (91) for the forces on the ellipsoid as a function of the number of evaluation points, for $n = 529$, 2025, 4624, and 8281. We note that for an ellipsoid in Eq. (88), the spherical harmonic interpolant

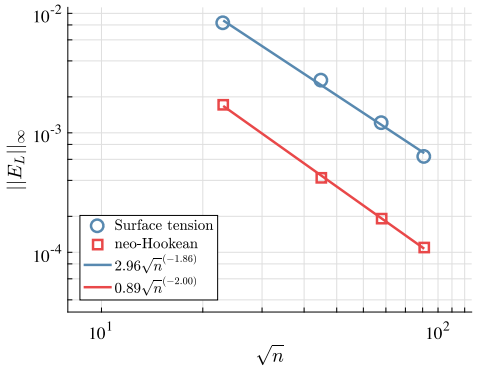
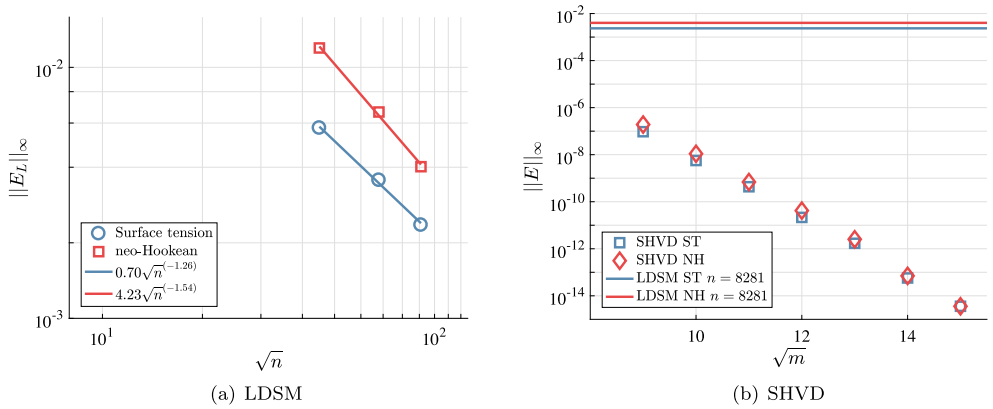


Fig. 2. Errors in surface tension and neo-Hookean forces on an ellipsoid. $\|\boldsymbol{E_L}\|$, the ℓ_∞ error in LDSM forces, is shown for the cases of 529, 2025, 4624, and 8281 points along with a power-law fit of the data. Because the spherical harmonic interpolant from Eq. (85) is exact for an ellipsoid, forces computed from SHVD are exact and $\|\boldsymbol{E_S}\| = 0$.



(a) LDSM (b) SHVD

Fig. 3. Errors in forces for the perturbed ellipsoid (Eq. (89)) for the two models. The discrete ℓ_∞ norms of $\boldsymbol{E_L}$ and $\boldsymbol{E_s}$ given by Eqs. (91) and (90) are graphed. In (a), the error $\|\boldsymbol{E_L}\|$ is plotted against the number of evaluation points n for both surface tension (blue circles) and neo-Hookean (red squares) forces. In (b), the error in SHVD $\|\boldsymbol{E_S}\|$ is plotted against the number of interpolation points m for both surface tension (blue squares) and neo-Hookean (red diamonds) forces. We observe spectral convergence in the number of interpolation points m. Forces computed with SHVD with as few as $m = 81$ interpolation points are several orders of magnitude more accurate than those computed from LDSM with $n = 8281$ evaluation points. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

with as little as 4 interpolation points gives the same mapping as Eq. (88). Therefore, the forces from SHVD are exact for an ellipsoid and are not included in Fig. 2. Consequently, we show in Fig. 2 only the errors in LDSM as a function of the number of evaluation points for both surface tension and neo-Hookean forces. A piecewise planar representation of a surface converges to a smooth surface with second order accuracy, so we expect at most second order accurate LDSM forces. We observe a second order convergence rate in the forces with respect to the length dimension (see power function fit of data in Fig. 2). We observe a similar convergence rate for computing surface tension on a sphere with LDSM (data not shown).

Our study of the perturbed ellipsoid shows that forces from SHVD converge to the exact solution with spectral accuracy in contrast to LDSM. Fig. 3 shows the errors in surface tension and neo-Hookean forces for LDSM with varying numbers of evaluation points n (Fig. 3(a)) and SHVD with varying numbers of interpolation points m (Fig. 3(b)). A convergence rate between first and second order is observed for LDSM. In contrast, the force from SHVD converges spectrally as the number of interpolation points m increases regardless of the number of evaluation points n [15]. The force estimate from SHVD is orders of magnitude smaller than LDSM when $m \geq 81$. This stems from the fact that SHVD relies on a *continuous* surface representation, and is therefore able to effectively capture the geometrical information of the surface necessary to accurately compute forces.

We comment that the error in forces from SHVD is from both interpolation error and aliasing, a well known phenomenon in which coefficients of higher order spherical harmonics are folded into lower order coefficients computed in Eq. (86) [41]. Here the analytical expressions for force are often not in the space of spherical harmonics of degree at most N, which makes them subject to aliasing error. Previous studies, among them [21,22], have accounted for this via upsampling of points for force calculations. We take a different approach by increasing the number of interpolation points until the error becomes

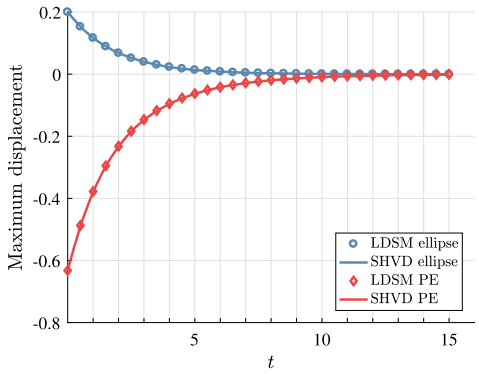


Fig. 4. Maximum displacement over time for dynamic simulations of the test objects. The displacement is the maximum displacement from the center of mass minus the radius in the reference configuration. Despite the inaccuracy of LDSM in computing the forces, the full IB simulation gives similar results to one that uses SHVD for force computations.

close to machine epsilon. Fig. 3(b) shows the error in the force calculation for the perturbed ellipse to be on the order 10^{-15} with 225 interpolation points used.

4.2. Dynamic test

We next perform full immersed boundary simulations on the test objects in Eqs. (88) and (89) with parameters $a = 1.2$, $b = c = 1/\sqrt{1.2}$, $B = 0.25$, $V_f \approx 5.14$, and $\sigma = K = A = G_s = 1$. Each object relaxes to its equilibrium configuration over time (in this case, the unit sphere). The relaxation timescale is dictated by the values of elastic moduli and the viscosity of the fluid, which we set to $\mu = 1$ for simplicity. We use 8281 evaluation points ($n = 8281$) for both objects and 225 interpolation points in the SHVD model ($m = 225$). From Fig. 2, this corresponds to an initial error in forces on the order of 10^{-3} for LDSM and 10^{-15} (machine epsilon) for SHVD. We use a total force that is the sum of surface tension and neo-Hookean forces (i.e. $\widehat{\boldsymbol{F}}_{TOT} = \widehat{\boldsymbol{F}}_{NH} + \widehat{\boldsymbol{F}}_{ST}$). The Eulerian grid runs from $a = -2$ to $b = 2$ in the x, y, and z directions with a grid size of $\eta = 32$ (32 Eulerian points in each direction). For each object, we track the maximum or minimum displacement from the center of mass as a function of time. Fig. 4 shows that, despite the large difference in accuracy between LDSM and SHVD force computations, both models give similar results for displacement over time due to the first order time update and the discrete delta function in the spreading and interpolation operators.

We study the convergence of each method by examining the state of the ellipsoid in Eq. (88) and Fig. 1(a) at $t = 3$ seconds. The initial surface configuration is the ellipsoid with $a = 1.2$, and $b = c = 1/\sqrt{1.2}$. We consider a fixed Eulerian grid of size η^3 Eulerian points. We then choose a Lagrangian grid with $n \approx 2\eta^2$ points. We let $\Delta t = 1/(2\eta)$ and measure the change in distance, denoted d_{max}, of the rightmost point (the point with largest x coordinate) minus the center of mass (i.e. it is the amount the rightmost point has moved in the x direction after 3 seconds after subtracting the center of mass). The error is the difference in d_{max} with respect to a grid for the same method that is 1.5 times as refined. For example, the error value for a 32 point LDSM grid is the change in x-displacement of the rightmost point from the center of mass on the 32 point grid minus the corresponding value on a 48 point grid, both using LDSM.

We consider grid sizes η, of 32, 48, and 64 corresponding to $n = 2025$, 4624, and 8281 evaluation points. For SHVD, we hold the number of interpolation points m constant at 64. Fig. 5 shows the results of the convergence test for the different grid sizes. Both methods display at least first order convergence in space and time. This is expected for the IB method, which as discussed in [42,43], is limited to first order accuracy in the case of an infinitely thin elastic membrane. We comment on this further in Section 6.

In addition to studying the convergence of each method under these circumstances, we also compare the computational costs. In a similar procedure to the convergence tests, we track the ratio of SHVD to LDSM execution times (i.e. time for SHVD / time for LDSM) for a 15 second ellipsoidal contraction with the same parameters as the convergence study. Table 1 summarizes the results. For m two or more orders of magnitude less than n, the wall clock time of our method is within 20% of LDSM. Observe that for larger m values, the $\mathcal{O}(mn)$ operations in our method's force computation increase the computational time farther beyond that of LDSM. We will see in Sections 5.1 and 5.2 how this behavior impacts the use of the model in our applications. For $m \ll n$ (the scenario under which our method is intended for use), the algorithms have comparable execution times.

5. Applications

In this section, we consider two different applications involving cellular dynamics: red blood cells in capillary flow and a model of bleb expansion. The former application involves large and smooth deformations of the entire cell surface due

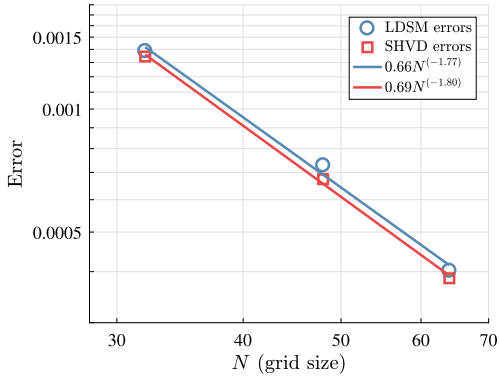


Fig. 5. Convergence tests on a dynamic simulation of an ellipsoid for both LDSM and SHVD. The error is measured as the change in displacement of the point with largest x coordinate relative to the center of mass, d_{max}. The error is the difference in d_{max} with respect to a grid 1.5 times as refined. Lines show power law function fits to the data which demonstrate the approximately first order accuracy of both methods.

Table 1
Ratio of computational times for 15 second ellipsoidal contraction.

η	n	t-ratio for $m = 64$	t-ratio for $m = 225$
32	2025	1.15	1.73
48	4624	1.08	1.46
64	8281	0.99	1.30
72	10404	1.00	1.32

to the background flow in a narrow tube. The membrane deformation in the model of cellular blebbing occurs because of a discontinuous applied force on a small region of the membrane surface. For both models, we compare simulation results from our method to LDSM. The results from the model simulations highlight the versatility of our approach.

5.1. Red blood cells in capillary flow

Red blood cells (RBCs) passing through microcapillaries is a well-studied biophysical problem that is important for understanding the rheological properties of both large and micro scale blood flow as well as transport via blood flow [44]. It has been observed experimentally that RBCs, which are nearly incompressible, deform into parachute-like shapes when flowing through capillaries, with the leading, or front, edge protruding under the influence of Poiseuille flow within the capillary [45]. For smaller tube diameters, it has been shown that RBCs align themselves in a single file configuration [45]. Our approach is therefore to model a single RBC flowing through a long capillary with SHVD. Because the velocity and length-scales are small, the Reynolds number for flow in the microcirculation is on the order $< 10^{-2}$, and the Stokes equations, Eqs. (1)–(2), govern the flow profile. We also simplify the problem by assuming the viscosity of the internal hemoglobin solution to be equal to that of the surrounding plasma [44].

5.1.1. Model of the RBC and capillary

The reference configuration of a red blood cell that we use is taken from [46]. Let $(x_s, y_s, z_s) = (\cos\lambda\cos\theta, \sin\lambda\cos\theta, \sin\theta)$ denote the mapping from $(\lambda, \theta) \to \mathbb{R}^3$ for the unit sphere. The RBC reference configuration is given by

$$\boldsymbol{Z_{RBC}} = R_{RBC} \begin{pmatrix} \frac{1}{2} x_s \left(0.21 + 2\left(y_s^2 + z_s^2\right) - 1.12\left(y_s^2 + z_s^2\right)^2 \right) \\ y_s \\ z_s \end{pmatrix}. \tag{92}$$

The mean radius of the RBC is $R_{RBC} = 3.91$ μm [44]. Elastic forces on the RBC come from resistance to stretching, shear, and bending. We use the neo-Hookean energy functional in Eq. (68) with $G_s = 2.5 \times 10^{-3}$ dyn/cm $= 2.5$ pN/μm and $A = 50$ to model resistance to shear and stretching. As discussed in [12,44], A is a computational parameter that is used to enforce the incompressibility of the RBC membrane while ensuring stability at large enough timesteps. [12] reported using A in the range from 8 to 400, and we found our value of $A = 50$ to give area deformations less than 1%. The derivatives of the reference configuration required for the neo-Hookean force calculation in Eq. (35) are computed analytically from the $\boldsymbol{Z_{RBC}}$ mapping. In addition, the area weights required to integrate the force over the RBC reference configuration are given by Eq. (51), where $\boldsymbol{Z}_{\mathbb{L}} = \boldsymbol{Z_{RBC}}$. For bending, we use the energy functional in Eq. (79) with a typical value $k_{bend} = 1 \times 10^{-12}$ erg $= 0.1$ pN·μm.

The capillary is modeled as a thin elastic cylindrical membrane of radius 5 μm that is discretized and tethered in place. Letting $\boldsymbol{Z}_{cap}$ and $\boldsymbol{X}_{cap}$ be the reference and deformed configurations of the capillary, the Lagrangian *force* on a capillary node is given by

$$\widehat{\boldsymbol{F}}_{teth}(\boldsymbol{q}) = -k_{teth}\left(\boldsymbol{X}_{cap}(\boldsymbol{q}) - \boldsymbol{Z}_{cap}(\boldsymbol{q})\right), \tag{93}$$

where $\boldsymbol{q}$ refers to the Lagrangian coordinate of the node and $k_{teth} = 2.45 \times 10^{-3}$ dyn/cm $= 2.45$ pN/μm. As discussed in [24], tether force functions such as Eq. (93) are not translation invariant, and are thus not guaranteed to integrate to zero in the discrete sense. Because the solution of the Stokes equations, Eqs. (1)–(2), on a periodic domain is non-unique, a constant velocity $\boldsymbol{u_c}(t)$ can be added to the structure velocities obtained from interpolation at each time step. Referring to Section 3.3, the time-stepping procedure is modified so that in the sixth step, the cylinder boundary is updated to an intermediate position

$$\hat{\boldsymbol{X}}_{cap}^{k+1} = \boldsymbol{X}_{cap}^{k} + \Delta t\, \boldsymbol{U}^{k+1}. \tag{94}$$

Then $\hat{\boldsymbol{X}}_{cap}^{k+1}$ is used to calculate a constant velocity that is added throughout the entire domain (also to the interpolated velocity of the RBC) to ensure that the tether forces sum to zero at the next time step. The constant velocity is given by

$$\boldsymbol{u_c}(t) = \frac{1}{A_{cap}\Delta t}\int_{cap}\left(\boldsymbol{Z}_{cap}(\boldsymbol{q}) - \hat{\boldsymbol{X}}_{cap}^{k+1}(\boldsymbol{q})\right) d\boldsymbol{q}, \tag{95}$$

where A_{cap} is the (lateral) surface area of the capillary. A derivation of Eq. (95) can be found in [24]. The positions of the Lagrangian structures (capillary and RBC) are then updated by $\boldsymbol{X}^{k+1} = \hat{\boldsymbol{X}}^{k+1} + \boldsymbol{u_c}(t)\Delta t$ to ensure the tether forces in Eq. (93) integrate to zero at the next time step.

A background force of 0.08 pN/μm$^3 = 8.0 \times 10^3$ dyn/cm^3 is specified inside the capillary to establish a parabolic flow profile as in [12]. Given the viscosity of blood plasma as 1.2 cP $= 1.2 \times 10^{-3}$ pN·s/μm^2, the resulting maximum flow velocity is 750 μm/s inside the capillary as in [12,45]. We impose a uniform force in the opposite direction outside of the capillary to ensure the integral of the background force over the Eulerian grid is zero. We use a 24 μm × 24 μm × 24 μm fluid grid. The long axis of the capillary is the x-axis, and the capillary runs from $x = -8$ to $x = 8$ μm. We center the cell at $(-8, 0, 0)$ initially and simulate its deformation until it reaches a steady state shape (around $t = 0.05$ s $= 500\Delta t$).

5.1.2. Results

As discussed in Section 2.6.3, there is no standard way to model bending energy in LDSM because the piecewise linear triangles have zero curvature. In addition, the bending force model from [12] is not guaranteed to integrate to zero in the discrete sense, meaning it cannot be used in a zero Reynolds number regime with periodic boundary conditions. Thus in order to fairly compare our model to LDSM, we begin by examining the case of $k_{bend} = 0$. For all simulations, we use $n = 8281$ evaluation points and vary the number of interpolation points m in our method.

Fig. 6 shows the deformation of the RBC from its biconcave initial shape into the parachute-like shapes observed in experimental capillary flow [45] for LDSM (top), and SHVD with $m = 625$ (middle) and $m = 400$ (bottom). We refer to the shape at $t = 0.05$ as the steady state shape of the RBC because it does not change significantly after this time value. While the steady state shapes are similar, a closer look reveals bumps in the RBCs modeled with LDSM and SHVD with $m = 625$. We take these asymmetric bumps to be non-physical, numerical error since the imposed flow and cylinder are symmetric. These bumps are distinct from the larger wavelength, axisymmetric, wrinkles observed in all of our simulations that exclude bending rigidity. We note that wrinkles have been observed in previous studies of shear flow at low bending rigidities [12,47]. When the number of interpolation points is greater than 625, we observe even more bumps on the cell (data not shown), indicating instability due to higher frequency modes in the interpolant. For the remainder of this section, we take $m = 400$ to obtain smooth surfaces similar to previous computational studies of RBCs [12,22].

Fig. 7 shows a comparison of the magnitude of the RBC elastic forces from simulations using LDSM and SHVD at $t = 0.05$. Although the overall accuracy in space and time of both methods is first order (see Section 4.2), the forces are much smoother with SHVD. Additionally, the magnitude of the forces is much larger in computations with LDSM than SHVD at the back end of the cell when its shape changes from concave to convex (and the mean curvature changes sign). The side views of the steady state shape (middle, Fig. 7) show wrinkles that occur in shapes from both simulations when the bending forces are excluded.

Surface bumps and wrinkles of the RBC are visible in Fig. 7 when $k_{bend} = 0$. Fig. 8 shows a head-on (facing the cell) view of the steady state cell shapes (at $t = 0.05$) for different values of bending rigidity. When $k_{bend} = 0$, the mean curvature H (defined in Eq. (76)) oscillates between positive and negative because of the wrinkling (Fig. 8, $k_{bend} = 0$). When $k_{bend} = 0.1$ pN·μm, surface wrinkles are significantly reduced, the curvature remains negative on the top face, and a smooth parachute shape is obtained. We note this is the same value of k_{bend} used in [12]. The curvature remains negative on the top face when $k_{bend} = 1$.

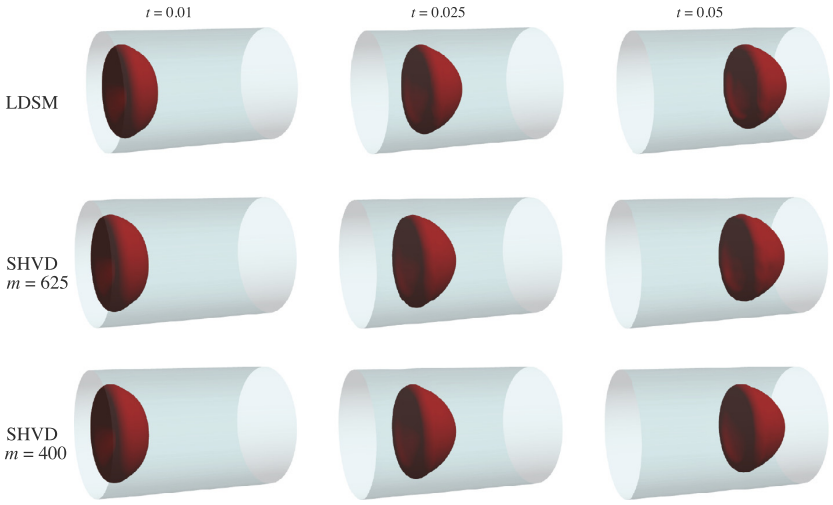


Fig. 6. The cell membrane and capillary at several time values using LDSM (top) and SHVD with $m = 625$ (middle) and $m = 400$ (bottom) interpolation points. All methods used $n = 8281$ evaluation points.

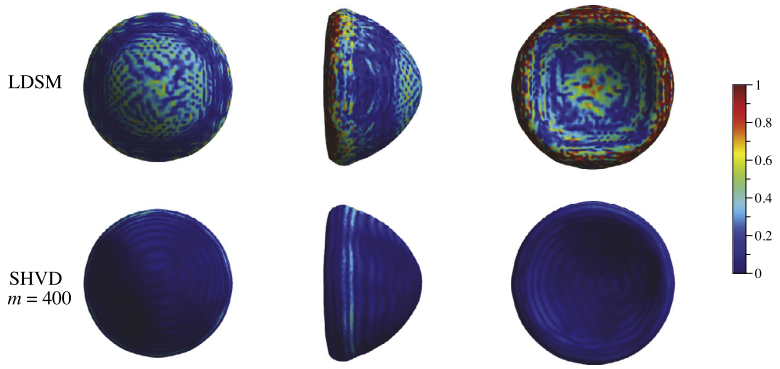


Fig. 7. Magnitude of the elastic forces on the cell at the front, side, and back views of the cell at $t = 0.05$ from simulations with LDSM (top) and SHVD with $m = 400$ interpolation points (bottom). $n = 8281$ evaluations points were used for both simulations.

At steady state, the front of cells with higher bending rigidities more closely resembles a sphere. We quantify this by examining the variability in mean curvature on the front edge of the cell as a function of the bending rigidity in Fig. 8. It can be seen that as k_{bend} increases from 0 to 1 pN·µm, the front of the cell more closely resembles a sphere with uniform curvature, as is expected for large bending rigidities. We note the smoothness of the curvature values in our method, which clearly shows the trend towards a more spherical shape for larger bending rigidities.

The bending rigidity is defined as the cell's resistance to being bent from a spherical shape with uniform curvature. Fig. 9 shows a slice of the RBC and capillary through the plane $y = 0$ at several time points during a simulation. The cell's deformation decreases as the value of bending rigidity increases. In particular, we observe that a cell with no bending rigidity initially deforms to become pinched at its center, then flat, and then curved. Meanwhile, the front edge of the cell maintains more uniform mean curvature throughout the simulation when $k_{bend} = 1$ pN·µm than with smaller values of k_{bend}.

5.2. Model of cellular blebbing

Blebs are round membrane protrusions that some cells use during migration [48]. Blebbing occurs when a thin layer of the cytoskeleton called the cortex delaminates from the cell membrane, where it is normally attached via linker (ERM) proteins [49]. Cells that bleb are usually under high cortical tension due to myosin molecular motors pulling actin filaments with respect to each other [50], a process referred to as actomyosin contractility. High actomyosin contractility within the cortex leads to high intracellular pressure (on the order of 10 Pa). When high intracellular pressure occurs along with a localized defect in the cortex or loss of membrane–cortex adhesion proteins, the membrane separates from the cortex, and

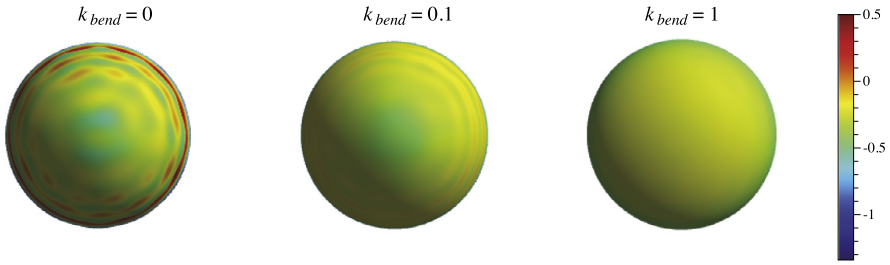


Fig. 8. Mean curvature of the cell viewed from the front for increasing values of k_{bend}. The cell was simulated with SHVD ($m = 400$ interpolation and $n = 8281$ evaluation points).

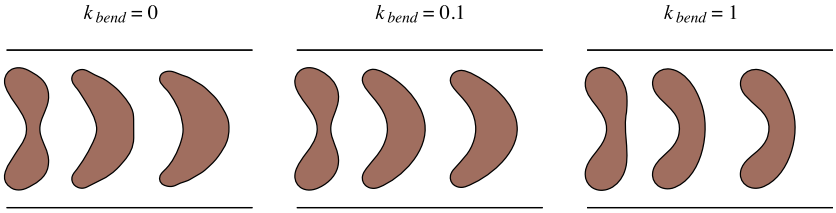


Fig. 9. Cross section of the cell and capillary tube through the plane $y = 0$ at several time values ($t = 0.005, 0.025$, and 0.05 from left to right) for when $k_{bend} = 0$ (left), 0.1 (middle), and 1 (right) pN·μm. The cell was simulated with SHVD ($m = 400$ interpolation and $n = 8281$ evaluation points).

cytoplasmic flow leads to bleb expansion. Hence, blebbing is a discontinuous deformation of a localized part of the cell in contrast to the global deformation of an RBC in a microcapillary.

Several mathematical models of cellular blebbing have been developed to understand the interactions between the cytoskeleton, cell membrane, and adhesion proteins. These models typically make simplifying assumptions, such as the system being in steady state [50,51], or are limited to two spatial dimensions [52,53]. In this section, we present a three-dimensional extension of the model from [52] as a first step towards understanding both the dynamics and geometry of these cellular protrusions. Our goal here is to present a dynamic 3D model of cellular blebbing where membrane forces are computed using both LDSM and SHVD. We compare bleb expansion dynamics and cell shapes generated by these methods. We also describe a modification to the existing method from [12,54] for computing mean curvature when using LDSM.

5.2.1. Mathematical model

Our mathematical model consists of the membrane, cortex, membrane–cortex adhesion, and intracellular fluid (see Fig. 10). The intracellular fluid represents the cytosol, the liquid part of the cytoplasm. The membrane is modeled as an impermeable elastic shell that experiences forces due to surface tension and stretching. The cortex is modeled as a permeable elastic structure, and membrane–cortex adhesion is modeled by elastic springs connecting the membrane to the cortex. The cytosol is modeled as a viscous incompressible fluid governed by the fluid equations

$$\begin{aligned} &\mu\Delta\boldsymbol{u} - \nabla p + \boldsymbol{f}_{\text{elastic}}^{\text{mem}} + \boldsymbol{f}_{\text{adh}}^{\text{mem/cortex}} + \boldsymbol{f}_{\text{drag}}^{\text{cortex}} = \boldsymbol{0} \\ &\nabla\cdot\boldsymbol{u} = 0, \end{aligned} \tag{96}$$

where the $\boldsymbol{f}_i$'s denote spread Lagrangian force densities due to membrane elasticity, membrane–cortex adhesion, and cortical drag. The drag force on the cortex is balanced by elastic forces within the cortex and adhesion to the membrane so that

$$\boldsymbol{F}_{\text{drag}}^{\text{cortex}} + \boldsymbol{F}_{\text{elastic}}^{\text{cortex}} + \boldsymbol{F}_{\text{adh}}^{\text{cortex/mem}} = \boldsymbol{0}. \tag{97}$$

The drag force density on the fluid is related to the cortex drag force density by

$$\boldsymbol{f}_{\text{drag}}^{\text{cortex}} = -\text{S}\boldsymbol{F}_{\text{drag}}^{\text{cortex}}, \tag{98}$$

where S denotes the spreading operator from Eq. (3).

Forces due to membrane elasticity are computed by either LDSM or SHVD with surface tension and neo-Hookean energy densities. The cortex is modeled by a different approach, similar to the lattice-spring model described in [5]. Specifically,

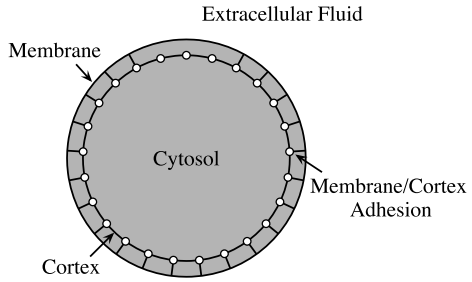


Fig. 10. Bleb model schematic as a slice of the cell through the *xz*-plane. A bleb is initiated by removing adhesive links in a small region at the top of the cell.

the cortex is treated as a collection of elastic springs, where the force at the ith point is computed by

$$\boldsymbol{F}_i\, dA_i = \sum_j k_{ij}^{\text{cortex}} \left(\frac{\left\| \boldsymbol{X}_i^{\text{cortex}} - \boldsymbol{X}_j^{\text{cortex}} \right\|}{d\ell_{ij}} \right) \frac{\boldsymbol{X}_i^{\text{cortex}} - \boldsymbol{X}_j^{\text{cortex}}}{\left\| \boldsymbol{X}_i^{\text{cortex}} - \boldsymbol{X}_j^{\text{cortex}} \right\|}, \tag{99}$$

where dA_i is the reference area differential, $d\ell_{ij}$ is the reference length of the spring connecting the ith point to the jth point on the surface, and j is the index of all points connected to the ith point. Note the area weights are the same as those from LDSM in Eq. (67). The spring coefficient connecting the ith point to the jth point is calculated as

$$k_{ij}^{\text{cortex}} = \frac{8k_{ij}}{3d\ell_{ij}} \left(\frac{dA_i + dA_j}{2} \right), \tag{100}$$

as in the lattice-spring model from [5]. This particular model of the cortex was used so that cortical ablation experiments could easily be performed by setting $k_{ij}^{\text{cortex}} = 0$ in a localized region. We plan to explore this mechanism of bleb initiation in a future publication.

Membrane–cortex adhesion is modeled by elastic springs attaching the membrane to the cortex with a force density given by

$$\boldsymbol{F}_{\text{adh}}^{\text{mem/cortex}} = k_{\text{adh}}^{\text{mem/cortex}} \Big(\| \boldsymbol{X}_{\text{mem}} - \boldsymbol{X}_{\text{cortex}} \| \Big) \frac{\boldsymbol{X}_{\text{mem}} - \boldsymbol{X}_{\text{cortex}}}{\| \boldsymbol{X}_{\text{mem}} - \boldsymbol{X}_{\text{cortex}} \|}. \tag{101}$$

The membrane–cortex adhesion stiffness coefficient $k_{\text{adh}}^{\text{cortex/mem}}$ was chosen so that the velocity of the cortex is zero if no bleb is initiated. The adhesion force density on the membrane is the opposite of the corresponding force density on the cortex, with the proper scaling to ensure that the two forces balance,

$$\int_\Omega \mathrm{S} \boldsymbol{F}_{\text{adhesion}}^{\text{mem/cortex}}\, d\boldsymbol{x} + \int_\Omega \mathrm{S} \boldsymbol{F}_{\text{adhesion}}^{\text{cortex/mem}}\, d\boldsymbol{x} = 0. \tag{102}$$

Given the stiffness coefficient $k_{\text{adh}}^{\text{cortex/mem}}$, the corresponding stiffness coefficient for the cortex is obtained by $k_{\text{adh}}^{\text{mem/cortex}} = k_{\text{adh}}^{\text{cortex/mem}} dA_i^{\text{cortex}} / dA_i^{\text{mem}}$, where dA_i represents the area differential in reference coordinates. For the membrane, these dA_i values are given by Eq. (45) for SHVD and Eq. (67) for LDSM.

Given a configuration of the membrane and cortex, the Lagrangian force densities are computed, and the velocities of the fluid and cortex are obtained by solving Eqs. (96) and (97). The positions of the membrane and cortex are then updated with their respective velocities,

$$\frac{d\boldsymbol{X}_{\text{mem}}}{dt} = \mathrm{S}^* \boldsymbol{u} = \boldsymbol{U}, \tag{103}$$

$$\frac{d\boldsymbol{X}_{\text{cortex}}}{dt} = \frac{1}{\xi} \left(\boldsymbol{F}_{\text{elastic}}^{\text{cortex}} + \boldsymbol{F}_{\text{attach}}^{\text{cortex/mem}} \right) + \boldsymbol{U} = \boldsymbol{U}_{\text{cortex}}, \tag{104}$$

where ξ is the drag coefficient of the cortex. A bleb is initiated by removing adhesive links between the membrane and cortex in a small region given by the equation $\phi \leq 2\pi/25$ in spherical coordinates (see Fig. 10). Model parameters values are provided in Table 2.

The model is simulated on a box with volume 30^3 μm^3 and $\Delta x = \Delta y = \Delta z = 30/32$. The membrane and cortex are initialized to be spheres centered at the point $(15, 15, 15)$ with 4761 Lagrangian points, corresponding to approximately 2 Lagrangian points per Eulerian grid cube. For the SHVD model, we compute all solutions with 4761 evaluation points

Table 2
Parameters for the blebbing model.

Symbol	Quantity	Value	Source
r_{mem}	Cell radius	10 μm	[50]
r_{cortex}	Cortex radius	9.99 μm	[53]
γ_{mem}	Membrane surface tension	40 pN/μm	[50]
κ_E	Membrane bulk modulus	40 pN/μm	[53]
μ_E	Membrane shear modulus	40 pN/μm	[53]
k_{cortex}	Cortical stiffness coefficient (average value)	87 pN/μm^2	
$k_{\text{adh}}^{\text{mem/cortex}}$	Membrane/cortex adhesion stiffness coefficient	$3 \cdot 10^4$ pN/μm^3	
μ	Cytosolic viscosity	1 Pa-s	[49,50,55]
ξ	Cortical drag coefficient	10 pN-s/μm^3	[52]

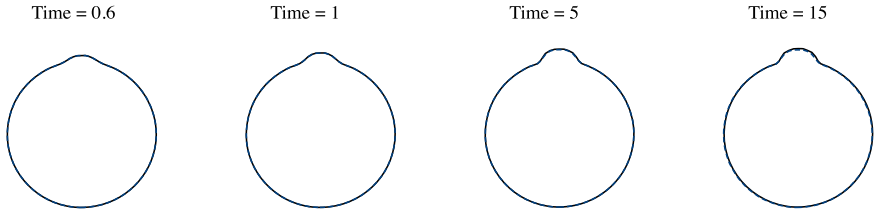


Fig. 11. Slices through the cell membrane by the yz plane when $x = 15$ at several time values. The black line denotes the membrane position computed by LDSM, and the blue dashed line indicates the membrane position computed by SHVD with 1024 interpolation points and 4761 evaluation points.

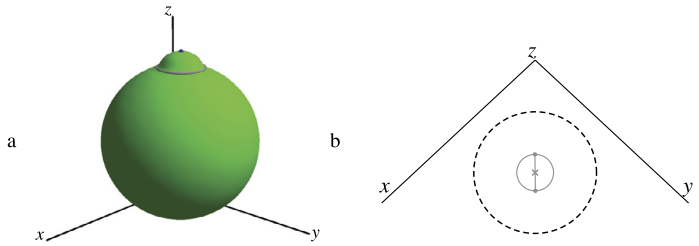


Fig. 12. The bleb ring is defined by the boundary between the region where adhesive links connect the membrane to the cortex and the region where the links are removed, indicated by the gray ring in (a). The membrane shell is graphed in green in (a). Bleb diameter is found by finding the point on the bleb ring that is closest to the point $(0, 0, 30)$ and the point directly across from it. A top town view of the cell and bleb ring is shown in (b) with the two points that define the bleb's diameter. The dashed black line indicates the membrane and the solid gray circle indicates the bleb ring. The distance from the midpoint of the bleb diameter (indicated by $\times$ in the top-down view) to the point with the largest z-coordinate value, as indicated by the blue point in (a), minus its initial value defines bleb size.

and vary the number of interpolation points to be 625, 1024, and 4761. The lattice spring model on the cortex results in large spurious velocities at the beginning of the simulation. Before initiating a bleb, we allow the forces in the model to equilibrate, meaning the velocity on the boundary decreases to approximately 0.01 μm/s. This process corresponds to 0.2 s of simulation time. Results are reported in time units after equilibration.

5.2.2. Model results

Pressure is initially uniform inside the cell. When adhesive links between the membrane and cortex are removed, cortical tension is no longer transmitted to the membrane locally, resulting in a pressure gradient and bleb expansion. Fig. 11 shows cross sections of the cell membrane as a bleb expands using both LSDM and SHVD with 1024 interpolation points. The cross sections show that the models have good agreement with each other.

In order to obtain a localized measurement of bleb size, we compute the bleb ring as the one dimensional boundary between the region where the membrane is attached to the cortex and the region where the membrane is detached from it (see Fig. 12). Bleb diameter is measured as the distance from the point closest to the z-axis to the point farthest away from it on the bleb ring. The midpoint of this line is computed, and bleb size is defined to be the vertical distance from the midpoint of the bleb diameter line to the point on the bleb with the largest z-coordinate (blue point in Fig. 12(a)) after subtracting the initial value. Note that this is a 3D extension of the method described in [53].

Bleb size over time when membrane forces are computed by LDSM and with several different values of interpolation points in SHVD is shown in Fig. 13. The data show very similar bleb expansion dynamics and steady state bleb size for all simulations. We observe that the bleb size increases as the bleb neck is better resolved with a larger number of

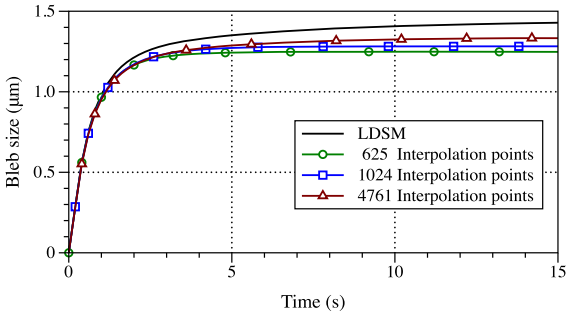


Fig. 13. Bleb size over time when the membrane position is computed by LDSM (black line) and by SHVD with various numbers of interpolation points and 4761 evaluation points.

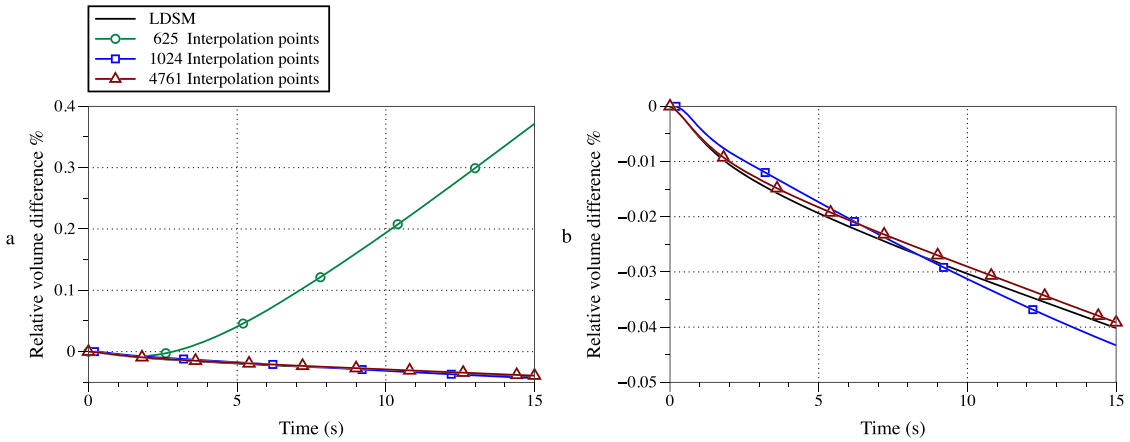


Fig. 14. Membrane volume over time when SHVD is used for the membrane with 625, 1024, and 4761 interpolation points. Data from the membrane modeled with LSDM are shown in black for comparison. The graph on the right is a zoomed in view of the same data from the left.

interpolation points. Once we reach $m = n = 4761$ (red triangles in Fig. 13), each evaluation is updated explicitly with the local fluid velocity. In this case (when $m = n$), the difference between our method and LDSM is due only to the method of computing forces.

Bleb volume over time is shown in Fig. 14, where percentage relative volume difference is defined as $(V(t) - V(0))/V(0) \times 100$, where $V(t)$ is the volume of the membrane at time t. Results from LDSM and SHVD with 1024 and 4761 interpolation points show good agreement. Volume actually increases over time with the SHVD model using 625 interpolation points. Because of the sharp transition from a region with membrane–cortex adhesion to one without adhesion, a large number of interpolation points is needed to resolve the region near the bleb neck.

Notably, the number of overall interpolation points required to resolve the bleb neck, which is a highly localized region of deformation, is higher than the number of points required to resolve the deformed RBC in Section 5.1 because the points overall must be evenly spread over the cell to yield an accurate interpolant. Hence more points must be added throughout the cell, not just in the localized region, in order to increase resolution of the bleb neck. Unlike our RBC application, adding points up to the limit $m = n$ actually *stabilizes* the steady state membrane shapes. We believe this is because most of the membrane remains still while tethered to the cortex. Both of these factors (the lack of motion from a background flow and the tether forces on the membrane) prevent higher order modes from distorting the membrane shape.

Membrane mean curvature after bleb expansion (at $t = 15$ s) is shown in Fig. 15 for the same data from Figs. 13 and 14. In order to compute curvature for LDSM, we follow the method described in [12,54], where curvature is computed on edges and vertices of surface triangles. This method results in curvatures that are always positive. In order to obtain signed mean curvature, we post-process the surface to compute its convex hull. Curvature values at points on the convex part of the surface are assigned to be negative while the rest of the curvature values (on the concave part of the surface) maintain their positive sign. In SHVD with 625 interpolation points, the bleb appears more broad which indicates it is not well-resolved. The membrane is effectively smoothed out if not enough interpolation points are used, and volume slightly increases. The membrane appears more dimpled in SHVD with 625 interpolation points compared to data from the other simulations. This can be partially attributed to the adhesion model, where membrane evaluation points are connected to

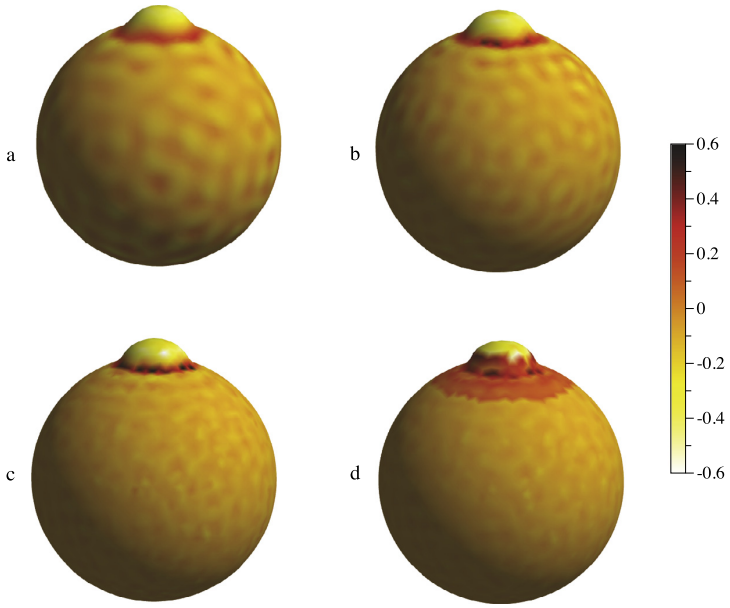


Fig. 15. Membrane mean curvature after bleb expansion at $t = 10$ s when the membrane is modeled using SHVD with (a) 625, (b) 1024, and (c) 4761 interpolation points. Mean curvature from LSDM with 4761 points is shown in (d).

points on the cortex. Since membrane evaluation points are not explicitly updated with the fluid velocity, adhesion forces are slightly perturbed when compared to model results from Fig. 15 (c) and (d), where the interpolation and evaluation points are updated with the local fluid velocity (although some dimpling is present even in Fig. 15 (c) and (d) due to membrane–cortex adhesion). The bleb appears to be well-resolved in SHVD with 1024 interpolation points with a smooth membrane. Membrane curvature from LSDM appears discontinuous and noisy when compared to simulation data from SHVD. We emphasize that SHVD more accurately computes curvature than LDSM given the ability of the spherical harmonic interpolant to capture the membrane shape.

The results from this section show that cell shape and bleb expansion dynamics are qualitatively similar when the membrane is modeled by LDSM and SHVD. If accurate geometric information, such as membrane curvature, is desired from a simulation, SHVD should be used for computing membrane forces. In [52], the authors used a 2D dynamic model to determine the role of cortical drag and cytoplasmic viscosity on bleb expansion dynamics. We find that for obtaining dynamic information such as bleb expansion time, LDSM is sufficiently accurate. In the next section, we comment on the computational cost associated with SHVD when compared to LDSM.

6. Discussion

We show that by representing surfaces with spherical harmonic interpolants, force densities can be computed more accurately as continuous functions by taking the variational derivative of an energy density functional than by discretizing the surface first into piecewise planar elements. Furthermore, geometric information such as curvature is more accurately computed with the continuous representation. We presented tests and applications for zero Reynolds number flow, but it is straightforward to extend our method to fluid–structure interaction problems where the fluid is governed by the Navier–Stokes equations. This is an advantage of our method compared to boundary integral methods that employ variational methods for computing forces, in that ours can also be used when modeling red blood cells in intermediate Reynolds number flows, for example [12].

A limitation of our method, as with all IB methods, is its overall first order accuracy in space because the regularized delta function used in the spreading operator does not accurately capture discontinuities in the pressure and derivatives of the velocity across the infinitely thin interfaces. Immersed interface methods [56] have been proposed to improve the accuracy of the IB method. These “IM methods” depend on quantifying the jump conditions at the interface, which are functions of geometric information, Lagrangian forces, and their derivatives [57]. The order of the method depends in part on how accurately these two quantities are computed. Because our method is spectrally accurate in both, a future goal of ours is to implement it in an IM framework, thereby increasing the overall accuracy of the method.

We also provide a method that ensures the integral of the Lagrangian force density is equal to the integral of the spread force density. This allows us to apply our method to models of structures immersed in Stokes flow, a regime that is prevalent across most cellular-level applications. We do note that our formulation relies on the structure being a closed surface.

Our method is applied to models of red blood cells in capillary flow and cellular blebbing. We have intentionally chosen two applications with very different dynamics and behavior in order to showcase the versatility of our method and highlight its strengths and limitations. In the RBC model, we see a smooth change in the entire surface over time. This allows us to resolve the shape with $\mathcal{O}(100)$ interpolation points. Lower numbers of interpolation points do not resolve the shape well enough, and higher numbers lead to numerical instabilities as higher frequency modes amplify non-physical bumps and noise on the cell surface.

Meanwhile, cellular blebbing involves a discontinuous change in a small region of the cell due to a discontinuity in the membrane–cortex adhesion force density. In our present formulation, the points must be evenly distributed to build a reliable interpolant and a large number of points must be used on the entire cell to accurately resolve the bleb neck. Therefore it is necessary to choose m (number of interpolation points) and n (number of evaluation points) to be on the same order of magnitude. We found that measuring volume of the membrane over time is an indicator of surface resolution; if membrane volume increases during a simulation, the surface is likely under-resolved and more interpolation points are needed. We hypothesize that the physical linkage of the membrane to the cortex stabilizes the higher frequency modes in the spherical harmonic interpolant, and numerical instabilities from a large number of interpolation points are not observed in this model.

The restriction $m \ll n$ is for computational efficiency; our method is most efficient if the $\mathcal{O}(mn)$ cost of computing the forces is comparable to the $\mathcal{O}(n)$ cost in LDSM. This is the case for the RBC model, where the surface is well-resolved with $\mathcal{O}(100)$ interpolation points and the computational time of our method is of the same order of magnitude as LDSM (approximately 1 minute). However, to simulate the blebbing model, m must be much larger, and the computational time to compute the forces increases significantly. A simulation of bleb expansion took approximately 10 hours with SHVD compared to approximately 1 hour for LDSM. While our model is versatile enough to be applied across a wide variety of applications, this additional cost should be taken into account.

We plan to use these principles to guide our future work. For example, our method would be well-suited for simulating a model of cytokinesis, the division of the cytoplasm when one cell divides into two cells [58]. During cytokinesis, the entire membrane deforms, and our method allows the force density to be computed at any point on the surface. Models of microtubule attachment to the cortex during spindle positioning will benefit from knowing the elastic force exactly at the points where the microtubules attach to the cortex. The noisy force data from LDSM could lead to inaccurate results. Finally, some models assume that spontaneous membrane curvature initializes furrow ingression during cytokinesis [59]. Our framework provides an accurate method for computing curvature in a dynamic 3D model of the early stages of cytokinesis.

Other future work includes extending our framework to include adaptive meshing by adding interpolation points when the surface shape undergoes large local deformations and surface features become unresolved. This capability, along with finding a more efficient way to compute the force densities (e.g. in parallel), could make our method more suitable for use with the blebbing model and allow us to investigate different bleb initiation mechanisms using our method. The formulation we have outlined can be applied to other interpolating basis functions, including radial basis functions. Because spherical harmonic basis functions were able to provide exact force calculations for very smooth (spherical) shapes, our method was computationally advantageous when it was applied to shapes resembling spheres. We anticipate that accuracy for arbitrary shapes (including blebbing cells) could be achieved by shifting the basis functions according to the application. We plan to investigate these topics in future publications.

Acknowledgements

This work was supported in part by grant #429808 from the Simons Foundation to W.S. A.K. was supported by a National Science Foundation grant DMS-1521748. The authors would like to thank Grady B. Wright for discussions about quadrature weights on the sphere, Varun Shankar for helpful discussions about surface representation, and Boyce Griffith for suggestions about computing forces due to bending. The authors also thank Thomas G. Fai and Charles S. Peskin for notes about modeling neo-Hookean materials in curvilinear coordinates.

Appendix A. Derivation of force density, $\boldsymbol{F}$

Here we derive in detail the force density in Eq. (35) from the variational derivative of the energy density functional. We make use of dyad expansions of tensors ($\boldsymbol{e_i e_j}$ denotes the ith row and jth column of the given tensor) and the definitions of the dot and double dot products ($\boldsymbol{e_i} \cdot \boldsymbol{e_j} = 1$ if $i = j$ and 0 otherwise, $\boldsymbol{e_i e_j} : \boldsymbol{e_k e_p} = 1$ if $j = k$ and $i = p$ and 0 otherwise).

A.1. Dyad expansion of $\mathcal{P}$ allows us to recover Eq. (21) from Eq. (23)

We begin with showing the equivalence of Eq. (21) and Eq. (23). In Eq. (21), the variational derivative is given by

$$\frac{\delta \mathcal{E}}{\delta \boldsymbol{X}} = -\sum_{i=1}^{3}\sum_{j=1}^{2} \frac{1}{\sqrt{\det \mathcal{G}_0}} \frac{\partial (\mathcal{P}_{ij}\sqrt{\det \mathcal{G}_0})}{\partial q_j}. \tag{A.1}$$

Considering the product in Eq. (23),

$$-\nabla_\theta \cdot \mathcal{P}^{\mathrm{T}} = -\frac{1}{\sqrt{\det \mathcal{G}_0}}\left(\frac{\partial}{\partial\lambda}\sqrt{\det \mathcal{G}_0}\,\boldsymbol{e_1} + \frac{\partial}{\partial\theta}\sqrt{\det \mathcal{G}_0}\,\boldsymbol{e_2}\right)\cdot$$

$$(\mathcal{P}_{11}\boldsymbol{e_1}\boldsymbol{e_1} + \mathcal{P}_{21}\boldsymbol{e_1}\boldsymbol{e_2} + \mathcal{P}_{31}\boldsymbol{e_1}\boldsymbol{e_3} + \mathcal{P}_{12}\boldsymbol{e_2}\boldsymbol{e_1} + \mathcal{P}_{22}\boldsymbol{e_2}\boldsymbol{e_2} + \mathcal{P}_{32}\boldsymbol{e_2}\boldsymbol{e_3}) \tag{A.2}$$

$$= -\sum_{i=1}^{3}\frac{1}{\sqrt{\det \mathcal{G}_0}}\left(\frac{\partial}{\partial\lambda}\left(\mathcal{P}_{i1}\sqrt{\det \mathcal{G}_0}\right) + \frac{\partial}{\partial\theta}\left(\mathcal{P}_{i2}\sqrt{\det \mathcal{G}_0}\right)\right) \tag{A.3}$$

$$= -\sum_{i=1}^{3}\sum_{j=1}^{2}\frac{1}{\sqrt{\det \mathcal{G}_0}}\left(\frac{\partial}{\partial q_j}\left(\mathcal{P}_{ij}\sqrt{\det \mathcal{G}_0}\right)\right). \tag{A.4}$$

Eq. (A.4) is the same expression as that derived from the variational derivative definition in Eq. (21),

$$\frac{\delta \mathcal{E}}{\delta \boldsymbol{X}} = -\nabla_\theta \cdot \mathcal{P}^{\mathrm{T}}. \tag{A.5}$$

A.2. Chain rule expansion of $\mathcal{P}^T$

We here prove Eq. (26),

$$\mathcal{P}^{\mathrm{T}} = \frac{\partial W}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}} = \frac{\partial W}{\partial \gamma} : \frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}} = \mathcal{S} : \frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}} = \mathcal{S}\cdot(\nabla_\theta \mathbf{X})^{\mathrm{T}}. \tag{A.6}$$

We begin with the double dot product $\frac{\partial W}{\partial \gamma} : \frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}}$. Recall from Eq. (14) that γ is the Green–Lagrange strain energy tensor, given by $\frac{1}{2}(\mathcal{G} - \mathcal{G}_0)$. Because γ is symmetric, the tensor $\frac{\partial W}{\partial \gamma} = \mathcal{S}$ (the second Piola–Kirchhoff stress tensor) is also symmetric, and its dyad expansion is

$$\frac{\partial W}{\partial \gamma} = \mathcal{S} = \mathcal{S}_{11}\boldsymbol{e_1}\boldsymbol{e_1} + \mathcal{S}_{12}\boldsymbol{e_1}\boldsymbol{e_2} + \mathcal{S}_{21}\boldsymbol{e_2}\boldsymbol{e_1} + \mathcal{S}_{22}\boldsymbol{e_2}\boldsymbol{e_2}. \tag{A.7}$$

Now, consider the fourth order tensor $\frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}}$. We denote

$$\left(\frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}}\right)_{ijk\ell} = \frac{\partial \gamma_{ij}}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}_{k\ell}}.$$

We write γ as

$$\gamma = \frac{1}{2}\begin{pmatrix} \frac{\partial \boldsymbol{X}}{\partial\lambda}\cdot\frac{\partial \boldsymbol{X}}{\partial\lambda} - \frac{\partial \boldsymbol{Z}}{\partial\lambda}\cdot\frac{\partial \boldsymbol{Z}}{\partial\lambda} & \frac{\partial \boldsymbol{X}}{\partial\lambda}\cdot\frac{\partial \boldsymbol{X}}{\partial\theta} - \frac{\partial \boldsymbol{Z}}{\partial\lambda}\cdot\frac{\partial \boldsymbol{Z}}{\partial\theta} \\ \frac{\partial \boldsymbol{X}}{\partial\theta}\cdot\frac{\partial \boldsymbol{X}}{\partial\lambda} - \frac{\partial \boldsymbol{Z}}{\partial\theta}\cdot\frac{\partial \boldsymbol{Z}}{\partial\lambda} & \frac{\partial \boldsymbol{X}}{\partial\theta}\cdot\frac{\partial \boldsymbol{X}}{\partial\theta} - \frac{\partial \boldsymbol{Z}}{\partial\theta}\cdot\frac{\partial \boldsymbol{Z}}{\partial\theta} \end{pmatrix}. \tag{A.8}$$

Expanding entrywise,

$$\gamma = \frac{1}{2}\begin{pmatrix} \frac{\partial X_1^2}{\partial\lambda} + \frac{\partial X_2^2}{\partial\lambda} + \frac{\partial X_3^2}{\partial\lambda} - \dots & \frac{\partial X_1}{\partial\lambda}\frac{\partial X_1}{\partial\theta} + \frac{\partial X_2}{\partial\lambda}\frac{\partial X_2}{\partial\theta} + \frac{\partial X_3}{\partial\lambda}\frac{\partial X_3}{\partial\theta} - \dots \\ \frac{\partial X_1}{\partial\lambda}\frac{\partial X_1}{\partial\theta} + \frac{\partial X_2}{\partial\lambda}\frac{\partial X_2}{\partial\theta} + \frac{\partial X_3}{\partial\lambda}\frac{\partial X_3}{\partial\theta} - \dots & \frac{\partial X_1^2}{\partial\theta} + \frac{\partial X_2^2}{\partial\theta} + \frac{\partial X_3^2}{\partial\theta} - \dots \end{pmatrix}. \tag{A.9}$$

Eq. (A.9) allows us to easily compute derivatives with respect to the entries of $(\nabla_\theta \mathbf{X})^{\mathrm{T}}$. For example,

$$\left(\frac{\partial \gamma}{\partial(\nabla_\theta \mathbf{X})^{\mathrm{T}}}\right)_{ij1\ell} = \frac{\partial \gamma}{\partial\left(\frac{\partial X_\ell}{\partial\lambda}\right)} = \begin{pmatrix} \frac{\partial X_\ell}{\partial\lambda} & \frac{1}{2}\frac{\partial X_\ell}{\partial\theta} \\ \frac{1}{2}\frac{\partial X_\ell}{\partial\theta} & 0 \end{pmatrix} \tag{A.10}$$

and

$$\left(\frac{\partial\gamma}{\partial(\nabla_\theta\mathbf{X})^{\mathrm{T}}}\right)_{ij2\ell}=\frac{\partial\gamma}{\partial\left(\frac{\partial X_\ell}{\partial\theta}\right)}=\begin{pmatrix}0 & \frac{1}{2}\frac{\partial X_\ell}{\partial\lambda}\\ \frac{1}{2}\frac{\partial X_\ell}{\partial\lambda} & \frac{\partial X_\ell}{\partial\theta}\end{pmatrix}. \tag{A.11}$$

The full expansion is

$$\frac{\partial W}{\partial\gamma}:\frac{\partial\gamma}{\partial(\nabla_\theta\mathbf{X})^{\mathrm{T}}}=\mathcal{S}:\frac{\partial\gamma}{\partial(\nabla_\theta\mathbf{X})^{\mathrm{T}}}=\sum_{i,j}\mathcal{S}_{ij}\boldsymbol{e_i}\boldsymbol{e_j}:\sum_{m,p,k,\ell}\left(\frac{\partial\gamma}{\partial(\nabla_\theta\mathbf{X})^{\mathrm{T}}}\right)_{mpk\ell}\boldsymbol{e_m}\boldsymbol{e_p}\boldsymbol{e_k}\boldsymbol{e_\ell}. \tag{A.12}$$

Recall that $\boldsymbol{e_i}\boldsymbol{e_j}:\boldsymbol{e_m}\boldsymbol{e_p}=1$ if $j=m$ and $i=p$ and is 0 otherwise. This reduces the (k,ℓ) entry of Eq. (A.12) to

$$\left(\mathcal{S}:\frac{\partial\gamma}{\partial(\nabla_\theta\mathbf{X})^{\mathrm{T}}}\right)_{k\ell}=\sum_{i,j}\mathcal{S}_{ij}\left(\frac{\partial\gamma}{\partial(\nabla_\theta\mathbf{X})^{\mathrm{T}}}\right)_{jik\ell}, \tag{A.13}$$

or in matrix form,

$$\mathcal{S}:\frac{\partial\gamma}{\partial(\nabla_\theta\mathbf{X})^{\mathrm{T}}}= \tag{A.14}$$
$$\begin{pmatrix}\mathcal{S}_{11}\frac{\partial X_1}{\partial\lambda}+\frac{1}{2}\mathcal{S}_{12}\frac{\partial X_1}{\partial\theta}+\frac{1}{2}\mathcal{S}_{21}\frac{\partial X_1}{\partial\theta} & \mathcal{S}_{11}\frac{\partial X_2}{\partial\lambda}+\frac{1}{2}\mathcal{S}_{12}\frac{\partial X_2}{\partial\theta}+\frac{1}{2}\mathcal{S}_{21}\frac{\partial X_2}{\partial\theta} & \mathcal{S}_{11}\frac{\partial X_3}{\partial\lambda}+\frac{1}{2}\mathcal{S}_{12}\frac{\partial X_3}{\partial\theta}+\frac{1}{2}\mathcal{S}_{21}\frac{\partial X_3}{\partial\theta}\\ \frac{1}{2}\mathcal{S}_{12}\frac{\partial X_1}{\partial\lambda}+\frac{1}{2}\mathcal{S}_{21}\frac{\partial X_1}{\partial\lambda}+\mathcal{S}_{22}\frac{\partial X_1}{\partial\theta} & \frac{1}{2}\mathcal{S}_{12}\frac{\partial X_2}{\partial\lambda}+\frac{1}{2}\mathcal{S}_{21}\frac{\partial X_2}{\partial\lambda}+\mathcal{S}_{22}\frac{\partial X_2}{\partial\theta} & \frac{1}{2}\mathcal{S}_{12}\frac{\partial X_3}{\partial\lambda}+\frac{1}{2}\mathcal{S}_{21}\frac{\partial X_3}{\partial\lambda}+\mathcal{S}_{22}\frac{\partial X_3}{\partial\theta}\end{pmatrix}.$$

By the symmetry of $\mathcal{S}$, we have

$$\mathcal{S}:\frac{\partial\gamma}{\partial(\nabla_\theta\mathbf{X})^{\mathrm{T}}}=\begin{pmatrix}\mathcal{S}_{11}\frac{\partial X_1}{\partial\lambda}+\mathcal{S}_{12}\frac{\partial X_1}{\partial\theta} & \mathcal{S}_{11}\frac{\partial X_2}{\partial\lambda}+\mathcal{S}_{12}\frac{\partial X_2}{\partial\theta} & \mathcal{S}_{11}\frac{\partial X_3}{\partial\lambda}+\mathcal{S}_{12}\frac{\partial X_3}{\partial\theta}\\ \mathcal{S}_{21}\frac{\partial X_1}{\partial\lambda}+\mathcal{S}_{22}\frac{\partial X_1}{\partial\theta} & \mathcal{S}_{21}\frac{\partial X_2}{\partial\lambda}+\mathcal{S}_{22}\frac{\partial X_2}{\partial\theta} & \mathcal{S}_{21}\frac{\partial X_3}{\partial\lambda}+\mathcal{S}_{22}\frac{\partial X_3}{\partial\theta}\end{pmatrix}. \tag{A.15}$$

From the dot product,

$$\mathcal{S}\cdot(\nabla_\theta\mathbf{X})^{\mathrm{T}}=(\mathcal{S}_{11}\boldsymbol{e_1}\boldsymbol{e_1}+\mathcal{S}_{12}\boldsymbol{e_1}\boldsymbol{e_2}+\mathcal{S}_{21}\boldsymbol{e_2}\boldsymbol{e_1}+\mathcal{S}_{22}\boldsymbol{e_2}\boldsymbol{e_2})\cdot$$
$$\left(\frac{\partial X_1}{\partial\lambda}\boldsymbol{e_1}\boldsymbol{e_1}+\frac{\partial X_2}{\partial\lambda}\boldsymbol{e_1}\boldsymbol{e_2}+\frac{\partial X_3}{\partial\lambda}\boldsymbol{e_1}\boldsymbol{e_3}+\frac{\partial X_1}{\partial\theta}\boldsymbol{e_2}\boldsymbol{e_1}+\frac{\partial X_2}{\partial\theta}\boldsymbol{e_2}\boldsymbol{e_2}+\frac{\partial X_3}{\partial\theta}\boldsymbol{e_2}\boldsymbol{e_3}\right) \tag{A.16}$$
$$=\begin{pmatrix}\mathcal{S}_{11}\frac{\partial X_1}{\partial\lambda}+\mathcal{S}_{12}\frac{\partial X_1}{\partial\theta} & \mathcal{S}_{11}\frac{\partial X_2}{\partial\lambda}+\mathcal{S}_{12}\frac{\partial X_2}{\partial\theta} & \mathcal{S}_{11}\frac{\partial X_3}{\partial\lambda}+\mathcal{S}_{12}\frac{\partial X_3}{\partial\theta}\\ \mathcal{S}_{21}\frac{\partial X_1}{\partial\lambda}+\mathcal{S}_{22}\frac{\partial X_1}{\partial\theta} & \mathcal{S}_{21}\frac{\partial X_2}{\partial\lambda}+\mathcal{S}_{22}\frac{\partial X_2}{\partial\theta} & \mathcal{S}_{21}\frac{\partial X_3}{\partial\lambda}+\mathcal{S}_{22}\frac{\partial X_3}{\partial\theta}\end{pmatrix} \tag{A.17}$$
$$=\mathcal{S}:\frac{\partial\gamma}{\partial(\nabla_\theta\mathbf{X})^{\mathrm{T}}}, \tag{A.18}$$

so that we recover Eq. (26).

A.3. Chain rule expansion of $\mathcal{S}$

As observed in Eq. (28), the second Piola–Kirchhoff stress tensor $\mathcal{S}$ can be computed by observing that $\mathcal{C}=\mathcal{I}_2+2\gamma\mathcal{G}_0^{-1}$, where $\mathcal{I}_2$ is the rank 2 identity tensor. Applying the chain rule twice yields

$$\mathcal{S}=\frac{\partial W}{\partial\gamma}=\frac{\partial W}{\partial\mathcal{C}}:\frac{\partial\mathcal{C}}{\partial\gamma}=2\frac{\partial W}{\partial\mathcal{C}}\cdot\mathcal{G}_0^{-1}, \tag{A.19}$$

and

$$\frac{\partial W}{\partial\mathcal{C}}=\frac{\partial W}{\partial I_1}\frac{\partial I_1}{\partial\mathcal{C}}+\frac{\partial W}{\partial I_2}\frac{\partial I_2}{\partial\mathcal{C}}. \tag{A.20}$$

The rank 2 tensor $\frac{\partial I_j}{\partial \mathcal{C}}$ is obtained by taking the derivative of the invariant I_j with respect to each element of $\mathcal{C}$. Carrying out these calculations,

$$I_1 = \mathrm{tr}\,\mathcal{C} - 2 = \mathcal{C}_{11} + \mathcal{C}_{22} - 2, \tag{A.21}$$

$$\begin{aligned} &\frac{\partial I_1}{\partial \mathcal{C}_{11}} = 1, \quad \frac{\partial I_1}{\partial \mathcal{C}_{12}} = 0, \\ &\frac{\partial I_1}{\partial \mathcal{C}_{21}} = 0, \quad \frac{\partial I_1}{\partial \mathcal{C}_{22}} = 1, \end{aligned} \tag{A.22}$$

$$\frac{\partial I_1}{\partial \mathcal{C}} = \mathcal{I}_2, \tag{A.23}$$

$$I_2 = \det \mathcal{C} - 1 = \mathcal{C}_{11}\mathcal{C}_{22} - \mathcal{C}_{12}\mathcal{C}_{21} - 1, \tag{A.24}$$

$$\begin{aligned} &\frac{\partial I_2}{\partial \mathcal{C}_{11}} = \mathcal{C}_{22}, \quad \frac{\partial I_2}{\partial \mathcal{C}_{12}} = -\mathcal{C}_{21}, \\ &\frac{\partial I_2}{\partial \mathcal{C}_{21}} = -\mathcal{C}_{12}, \quad \frac{\partial I_2}{\partial \mathcal{C}_{22}} = \mathcal{C}_{11}, \end{aligned} \tag{A.25}$$

$$\frac{\partial I_2}{\partial \mathcal{C}} = (\det \mathcal{C})(\mathcal{C}^{-1})^{\mathrm{T}} = (\det \mathcal{C})\left(\left(\mathcal{G}\mathcal{G}_0^{-1}\right)^{-1}\right)^{\mathrm{T}} = (\det \mathcal{C})\left(\mathcal{G}_0\mathcal{G}^{-1}\right)^{\mathrm{T}} = (\det \mathcal{C})\left(\mathcal{G}^{-1}\mathcal{G}_0\right). \tag{A.26}$$

The last equality follows from the symmetry of $\mathcal{G}$ and $\mathcal{G}_0$. At last we have that

$$\mathcal{S} = 2\frac{\partial W}{\partial \mathcal{C}} \cdot \mathcal{G}_0^{-1} = 2\left(\frac{\partial W}{\partial I_1}\frac{\partial I_1}{\partial \mathcal{C}} + \frac{\partial W}{\partial I_2}\frac{\partial I_2}{\partial \mathcal{C}}\right) \cdot \mathcal{G}_0^{-1} \tag{A.27}$$

$$= 2\left(\frac{\partial W}{\partial I_1}\mathcal{I}_2 + \frac{\partial W}{\partial I_2}(\det \mathcal{C})\left(\mathcal{G}^{-1}\mathcal{G}_0\right)\right) \cdot \mathcal{G}_0^{-1} \tag{A.28}$$

$$\mathcal{S} = 2\frac{\partial W}{\partial I_1}\mathcal{G}_0^{-1} + 2\frac{\partial W}{\partial I_2}(\det \mathcal{C})\mathcal{G}^{-1}. \tag{A.29}$$

A.4. The force density

Computing the force density in Eq. (24), we use the chain rule expansions of $\mathcal{P}^{\mathrm{T}}$ and $\mathcal{S}$:

$$\nabla_\theta \cdot = \frac{1}{\sqrt{\det \mathcal{G}_0}}\left(\frac{\partial}{\partial \lambda}\sqrt{\det \mathcal{G}_0}\,\boldsymbol{e_1} + \frac{\partial}{\partial \theta}\sqrt{\det \mathcal{G}_0}\,\boldsymbol{e_2}\right)\cdot, \tag{A.30}$$

$$\boldsymbol{F} = \nabla_\theta \cdot \mathcal{P}^{\mathrm{T}} = \nabla_\theta \cdot \left(\mathcal{S} \cdot (\nabla_\theta \mathbf{X})^{\mathrm{T}}\right), \tag{A.31}$$

$$\begin{aligned} \boldsymbol{F} = \nabla_\theta \cdot \Bigg(&\left(\mathcal{S}_{11}\frac{\partial X_1}{\partial \lambda} + \mathcal{S}_{12}\frac{\partial X_1}{\partial \theta}\right)\boldsymbol{e_1}\boldsymbol{e_1} + \left(\mathcal{S}_{11}\frac{\partial X_2}{\partial \lambda} + \mathcal{S}_{12}\frac{\partial X_2}{\partial \theta}\right)\boldsymbol{e_1}\boldsymbol{e_2} \\ &+ \left(\mathcal{S}_{11}\frac{\partial X_3}{\partial \lambda} + \mathcal{S}_{12}\frac{\partial X_3}{\partial \theta}\right)\boldsymbol{e_1}\boldsymbol{e_3} + \left(\mathcal{S}_{21}\frac{\partial X_1}{\partial \lambda} + \mathcal{S}_{22}\frac{\partial X_1}{\partial \theta}\right)\boldsymbol{e_2}\boldsymbol{e_1} \\ &+ \left(\mathcal{S}_{21}\frac{\partial X_2}{\partial \lambda} + \mathcal{S}_{22}\frac{\partial X_2}{\partial \theta}\right)\boldsymbol{e}_2\boldsymbol{e}_2 + \left(\mathcal{S}_{21}\frac{\partial X_3}{\partial \lambda} + \mathcal{S}_{22}\frac{\partial X_3}{\partial \theta}\right)\boldsymbol{e}_2\boldsymbol{e}_3\Bigg). \end{aligned} \tag{A.32}$$

Thus

$$\boldsymbol{F} = \frac{1}{\sqrt{\det \mathcal{G}_0}}\begin{pmatrix} \frac{\partial}{\partial \lambda}\sqrt{\det \mathcal{G}_0}\left(\mathcal{S}_{11}\frac{\partial X_1}{\partial \lambda} + \mathcal{S}_{12}\frac{\partial X_1}{\partial \theta}\right) + \frac{\partial}{\partial \theta}\sqrt{\det \mathcal{G}_0}\left(\mathcal{S}_{21}\frac{\partial X_1}{\partial \lambda} + \mathcal{S}_{22}\frac{\partial X_1}{\partial \theta}\right) \\ \frac{\partial}{\partial \lambda}\sqrt{\det \mathcal{G}_0}\left(\mathcal{S}_{11}\frac{\partial X_2}{\partial \lambda} + \mathcal{S}_{12}\frac{\partial X_2}{\partial \theta}\right) + \frac{\partial}{\partial \theta}\sqrt{\det \mathcal{G}_0}\left(\mathcal{S}_{21}\frac{\partial X_2}{\partial \lambda} + \mathcal{S}_{22}\frac{\partial X_2}{\partial \theta}\right) \\ \frac{\partial}{\partial \lambda}\sqrt{\det \mathcal{G}_0}\left(\mathcal{S}_{11}\frac{\partial X_3}{\partial \lambda} + \mathcal{S}_{12}\frac{\partial X_3}{\partial \theta}\right) + \frac{\partial}{\partial \theta}\sqrt{\det \mathcal{G}_0}\left(\mathcal{S}_{21}\frac{\partial X_3}{\partial \lambda} + \mathcal{S}_{22}\frac{\partial X_3}{\partial \theta}\right) \end{pmatrix} \tag{A.33}$$

$$\boldsymbol{F} = \frac{1}{\sqrt{\det \mathcal{G}_0}} \left(\frac{\partial}{\partial \lambda} \sqrt{\det \mathcal{G}_0} \left(\mathcal{S}_{11} \frac{\partial \boldsymbol{X}}{\partial \lambda} + \mathcal{S}_{12} \frac{\partial \boldsymbol{X}}{\partial \theta} \right) + \frac{\partial}{\partial \theta} \sqrt{\det \mathcal{G}_0} \left(\mathcal{S}_{21} \frac{\partial \boldsymbol{X}}{\partial \lambda} + \mathcal{S}_{22} \frac{\partial \boldsymbol{X}}{\partial \theta} \right) \right). \tag{A.34}$$

Eq. (A.34) recovers Eq. (35) from the main text.

Appendix B. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jcp.2018.05.045.

References

[1] C.S. Peskin, Flow patterns around heart valves: a numerical method, J. Comput. Phys. 10 (2) (1972) 252–271, https://doi.org/10.1016/0021-9991(72)90065-4.
[2] L.A. Miller, C.S. Peskin, A computational fluid dynamics of 'clap and fling' in the smallest insects, J. Exp. Biol. 208 (2) (2005) 195–212, https://doi.org/10.1242/jeb.01376.
[3] L.A. Miller, C.S. Peskin, When vortices stick: an aerodynamic transition in tiny insect flight, J. Exp. Biol. 207 (17) (2004) 3073–3088, https://doi.org/10.1242/jeb.01138.
[4] L.J. Fauci, C.S. Peskin, A computational model of aquatic animal locomotion, J. Comput. Phys. 77 (1) (1988) 85–108, https://doi.org/10.1016/0021-9991(88)90158-1.
[5] W. Strychalski, C.A. Copos, O.L. Lewis, R.D. Guy, A poroelastic immersed boundary method with applications to cell biology, J. Comput. Phys. 282 (2015) 77–97, https://doi.org/10.1016/j.jcp.2014.10.004.
[6] R.P. Beyer, A computational model of the cochlea using the immersed boundary method, J. Comput. Phys. 98 (1) (1992) 145–162, https://doi.org/10.1016/0021-9991(92)90180-7.
[7] B. Griffith, X. Luo, L.H. Charney, Hybrid finite difference/finite element version of the immersed boundary method, Int. J. Numer. Methods Biomed. Eng. (2017) e2888, https://doi.org/10.1002/cnm.2888.
[8] D. Boffi, L. Gastaldi, A finite element approach for the immersed boundary method, Comput. Struct. 81 (8) (2003) 491–501, https://doi.org/10.1016/S0045-7949(02)00404-2.
[9] D. Boffi, L. Gastaldi, L. Heltai, C.S. Peskin, On the hyper-elastic formulation of the immersed boundary method, Comput. Methods Appl. Math. 197 (25) (2008) 2210–2231, https://doi.org/10.1016/j.cma.2007.09.015.
[10] D. Boffi, L. Gastaldi, L. Heltai, On the CFL condition for the finite element immersed boundary method, Comput. Struct. 85 (11) (2007) 775–783, https://doi.org/10.1016/j.compstruc.2007.01.009.
[11] D. Devendran, C.S. Peskin, An immersed boundary energy-based method for incompressible viscoelasticity, J. Comput. Phys. 231 (14) (2012) 4613–4642, https://doi.org/10.1016/j.jcp.2012.02.020.
[12] T.G. Fai, B.E. Griffith, Y. Mori, C.S. Peskin, Immersed boundary method for variable viscosity and variable density problems using fast constant-coefficient linear solvers I: numerical method and results, SIAM J. Sci. Comput. 35 (5) (2013) B1132–B1161, https://doi.org/10.1137/120903038.
[13] C.-H. Wu, T.G. Fai, P.J. Atzberger, C.S. Peskin, Simulation of osmotic swelling by the stochastic immersed boundary method, SIAM J. Sci. Comput. 37 (4) (2015) B660–B688, https://doi.org/10.1137/14098404X.
[14] V. Camacho, A. Fogelson, J. Keener, Eulerian–Lagrangian treatment of nondilute two-phase gels, SIAM J. Appl. Math. 76 (1) (2016) 341–367, https://doi.org/10.1137/15M1023579.
[15] V. Shankar, G.B. Wright, A.L. Fogelson, R.M. Kirby, A study of different modeling choices for simulating platelets within the immersed boundary method, Appl. Numer. Math. 63 (2013) 58–77, https://doi.org/10.1016/j.apnum.2012.09.006.
[16] V. Shankar, G.B. Wright, R.M. Kirby, A.L. Fogelson, Augmenting the immersed boundary method with radial basis functions (RBFs) for the modeling of platelets in hemodynamic flows, Int. J. Numer. Methods Fluids 79 (10) (2015) 536–557, https://doi.org/10.1002/fld.4061.
[17] A. Kumar, M.D. Graham, Cell distribution and segregation phenomena during blood flow, in: Complex Fluids in Biological Systems, Springer, 2015, pp. 399–435.
[18] D. Barthès-Biesel, Motion and deformation of elastic capsules and vesicles in flow, Annu. Rev. Fluid Mech. 48 (2016) 25–52, https://doi.org/10.1146/annurev-fluid-122414-034345.
[19] C. Pozrikidis, Boundary Integral and Singularity Methods for Linearized Viscous Flow, Cambridge University Press, 1992.
[20] C. Pozrikidis, Interfacial dynamics for Stokes flow, J. Comput. Phys. 169 (2) (2001) 250–301, https://doi.org/10.1006/jcph.2000.6582.
[21] H. Zhao, A.H. Isfahani, L.N. Olson, J.B. Freund, A spectral boundary integral method for flowing blood cells, J. Comput. Phys. 229 (10) (2010) 3726–3744, https://doi.org/10.1016/j.jcp.2010.01.024.
[22] S.K. Veerapaneni, A. Rahimian, G. Biros, D. Zorin, A fast algorithm for simulating vesicle flows in three dimensions, J. Comput. Phys. 230 (14) (2011) 5610–5634, https://doi.org/10.1016/j.jcp.2011.03.045.
[23] G. Batchelor, An Introduction to Fluid Mechanics, Cambridge University Press, New York, 1967.
[24] J.M. Teran, C.S. Peskin, Tether force constraints in Stokes flow by the immersed boundary method on a periodic domain, SIAM J. Sci. Comput. 31 (5) (2009) 3404–3416, https://doi.org/10.1137/080720217.
[25] C.S. Peskin, The immersed boundary method, Acta Numer. 11 (2002) 479–517, https://doi.org/10.1017/S0962492902000077.
[26] M. Kraus, W. Wintz, U. Seifert, R. Lipowsky, Fluid vesicles in shear flow, Phys. Rev. Lett. 77 (17) (1996) 3685, https://doi.org/10.1103/PhysRevLett.77.3685.
[27] K.D. Hjelmstad, Fundamentals of Structural Mechanics, Springer Science & Business Media, 2007.
[28] R.W. Ogden, Non-Linear Elastic Deformations, Courier Corporation, 1997.
[29] M. Gräf, S. Kunis, D. Potts, On the computation of nonnegative quadrature weights on the sphere, Appl. Comput. Harmon. Anal. 27 (1) (2009) 124–132, https://doi.org/10.1016/j.acha.2008.12.003.
[30] D. Kim, S. Sra, I.S. Dhillon, A non-monotonic method for large-scale non-negative least squares, Optim. Methods Softw. 28 (5) (2013) 1012–1039, https://doi.org/10.1080/10556788.2012.656368.
[31] R.S. Womersley, I.H. Sloan, How good can polynomial interpolation on the sphere be?, Adv. Comput. Math. 14 (3) (2001) 195–226, https://doi.org/10.1023/A:1016630227163.
[32] I.H. Sloan, R.S. Womersley, Extremal systems of points and numerical integration on the sphere, Adv. Comput. Math. 21 (1–2) (2004) 107–125, https://doi.org/10.1023/B:ACOM.0000016428.25905.da.
[33] R. Womersley, http://web.maths.unsw.edu.au/~rsw/Sphere/Extremal/New/index.html#S:Vals.
[34] E. Fuselier, T. Hangelbroek, F.J. Narcowich, J.D. Ward, G.B. Wright, Kernel based quadrature on spheres and other homogeneous spaces, Numer. Math. 127 (1) (2014) 57–92, https://doi.org/10.1007/s00211-013-0581-1.
[35] E.A. Evans, R. Skalak, S. Weinbaum, Mechanics and Thermodynamics of Biomembranes, 1980.

[36] R. Capovilla, Elastic bending energy: a variational approach, J. Geom. Symmetry Phys. 45 (2017) 1–45.
[37] C. Pozrikidis, Modeling and Simulation of Capsules and Biological Cells, CRC Press, 2003.
[38] O.-Y. Zhong-Can, W. Helfrich, Bending energy of vesicle membranes: general expressions for the first, second, and third variation of the shape energy and applications to spheres and cylinders, Phys. Rev. A 39 (10) (1989) 5280, https://doi.org/10.1103/PhysRevA.39.5280.
[39] R. Capovilla, J. Guven, Second variation of the Helfrich–Canham Hamiltonian and reparametrization invariance, J. Phys. A, Math. Gen. 37 (23) (2004) 5983, https://doi.org/10.1088/0305-4470/37/23/003.
[40] R. Peyret, Spectral Methods for Incompressible Viscous Flow, Appl. Math. Sci., vol. 148, Springer, 2002.
[41] C. Jekeli, Spherical harmonic analysis, aliasing, and filtering, J. Geod. 70 (4) (1996) 214–223, https://doi.org/10.1007/BF00873702.
[42] M.-C. Lai, C.S. Peskin, An immersed boundary method with formal second-order accuracy and reduced numerical viscosity, J. Comput. Phys. 160 (2) (2000) 705–719, https://doi.org/10.1006/jcph.2000.6483.
[43] B.E. Griffith, C.S. Peskin, On the order of accuracy of the immersed boundary method: higher order convergence rates for sufficiently smooth problems, J. Comput. Phys. 208 (1) (2005) 75–105, https://doi.org/10.1016/j.jcp.2005.02.011.
[44] C. Pozrikidis, Axisymmetric motion of a file of red blood cells through capillaries, Phys. Fluids 17 (3) (2005) 031503, https://doi.org/10.1063/1.1830484.
[45] K. Tsukada, E. Sekizuka, C. Oshio, H. Minamitani, Direct measurement of erythrocyte deformability in diabetes mellitus with a transparent microchannel capillary model and high-speed video camera system, Microvasc. Res. 61 (3) (2001) 231–239, https://doi.org/10.1006/mvre.2001.2307.
[46] T. Omori, T. Ishikawa, D. Barthès-Biesel, A.-V. Salsac, Y. Imai, T. Yamaguchi, Tension of red blood cell membrane in simple shear flow, Phys. Rev. E 86 (5) (2012) 056321, https://doi.org/10.1103/PhysRevE.86.056321.
[47] A.Z. Yazdani, P. Bagchi, Phase diagram and breathing dynamics of a single red blood cell and a biconcave capsule in dilute shear flow, Phys. Rev. E 84 (2) (2011) 026314, https://doi.org/10.1103/PhysRevE.84.026314.
[48] G. Charras, E. Paluch, Blebs lead the way: how to migrate without lamellipodia, Nat. Rev. Mol. Cell Biol. 9 (9) (2008) 730–736, https://doi.org/10.1038/nrm2453.
[49] G.T. Charras, M. Coughlin, T.J. Mitchison, L. Mahadevan, Life and times of a cellular bleb, Biophys. J. 94 (5) (2008) 1836–1853, https://doi.org/10.1529/biophysj.107.113605.
[50] J.Y. Tinevez, U. Schulze, G. Salbreux, J. Roensch, J.F. Joanny, E. Paluch, Role of cortical tension in bleb growth, Proc. Natl. Acad. Sci. USA 106 (44) (2009) 18581–18586, https://doi.org/10.1073/pnas.0903353106.
[51] T.E. Woolley, E.A. Gaffney, J.M. Oliver, S.L. Waters, R.E. Baker, A. Goriely, Global contraction or local growth, bleb shape depends on more than just cell structure, J. Theor. Biol. 380 (7) (2015) 83–97, https://doi.org/10.1016/j.jtbi.2015.04.023.
[52] W. Strychalski, R.D. Guy, A computational model of bleb formation, Math. Med. Biol. 30 (2) (2013) 115–130, https://doi.org/10.1093/imammb/dqr030.
[53] W. Strychalski, R.D. Guy, Intracellular pressure dynamics in blebbing cells, Biophys. J. 110 (5) (2016) 1168–1179, https://doi.org/10.1016/j.bpj.2016.01.012.
[54] J.M. Sullivan, Curvatures of smooth and discrete surfaces, in: Discrete Differential Geometry, vol. 38, Springer, 2008, pp. 175–188.
[55] E. Moeendarbary, L. Valon, M. Fritzsche, A.R. Harris, D.A. Moulding, A.J. Thrasher, E. Stride, L. Mahadevan, G.T. Charras, The cytoplasm of living cells behaves as a poroelastic material, Nat. Mater. 12 (3) (2013) 253–261, https://doi.org/10.1038/nmat3517.
[56] Z. Li, K. Ito, The Immersed Interface Method: Numerical Solutions of PDEs Involving Interfaces and Irregular Domains, vol. 33, SIAM, 2006.
[57] S. Xu, Z.J. Wang, A 3D immersed interface method for fluid–solid interaction, Comput. Methods Appl. Mech. Eng. 197 (25–28) (2008) 2068–2086, https://doi.org/10.1016/j.cma.2007.06.012.
[58] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, Molecular Biology of the Cell, 4th edition, Garland Science, New York, 2002.
[59] R. Shlomovitz, N.S. Gov, Physical model of contractile ring initiation in dividing cells, Biophys. J. 94 (4) (2008) 1155–1168, https://doi.org/10.1529/biophysj.107.111351.

CHAPTER 4

PARALLELIZATION STRATEGIES FOR INTERPOLATION AND SPREADING

As the computing needs of the scientific community increase, so too must the speed at which computing resources solve their problems. Unfortunately, power consumption and temperature control hinder the development of ever-faster processors. Instead, the trend in computing is to use more, slower processors acting in concert. These processors can share a memory address space, or the memory can be distributed across multiple processes or devices, called nodes. Modern supercomputers, for example, are composed of many nodes, each with their own multicore central processing units (CPUs), and often attached to general-purpose graphical processing units (GPGPU; GPU for short).

To take advantage of modern computing architectures, with ever-increasing numbers of processors, it is necessary to develop parallel algorithms for the IB method. McQueen and Peskin [55] present a domain decomposition scheme to parallelize the interpolation and spreading operations on the Cray C-90 computer with shared memory and a modest number of vector processors. Their results illustrate the need for a fast interpolation and spreading: even parallelized, they spend roughly half of the wall clock time spreading and interpolating. Fai *et al.* adapted this domain decomposition scheme for use on a GPU [47]. Patel used a GPU to spread forces by processing all δ_h values in parallel, one Lagrangian point at a time [58]. Valero-Lara *et al.* accelerated their IB-Lattice Boltzmann code by processing Lagrangian points in parallel, and using atomic operations when necessary [62]. Because none of these approaches concurrently process arbitrary Lagrangian points, it is easy to find cases for which they perform poorly. An alternative is to distribute work among several devices, each with its own memory. This idea underpins the popular `IBAMR` library (see [48–52]), which also adaptively refines the mesh around the immersed structure. With proper load-balancing, a serial algorithm processes a smaller portion of work. A cluster of multicore devices, however, will not be used effectively without a shared memory parallel algorithm. The `cuIBM` [54] and `PetIBM` [46, 56] libraries implement an adaptive IB method for prescribed motion on single- and multi-GPU architectures, respectively. The authors

demonstrate their method on a few two-dimensional test problems. Their implementation explicitly constructs the spreading and interpolation operators, which are sparse. Effective parallelization methods for sparse matrix-vector multiplication rely on distributing equal work among the threads. Multiplication against banded matrices with an approximately equal number of nonzero entries per row, for example, are easily parallelized as a series of sparse dot products. The spreading operator has approximately the same number of nonzero entries per column, but this sparsity pattern does not always lend itself well to parallelization.

For this dissertation, our primary target is a single NVIDIA® GPU. While some GPUs now support a shared memory address space between the CPU and GPU, we reserve the CPU for input/output operations and trivial calculations. We will also defer discussing distributed memory parallelization to Section 6.2. GPUs have restrictions on their parallelization. They use single instruction, multiple data (SIMD) parallelism, in which each computational unit, or thread, executes the same instruction on its own data. Concurrency on the GPU is typically limited by the amount of shared memory, which is accessible to every thread in a group or "cooperative thread array", and register memory, which is available only to a single thread. The alternative is to use global memory, which is available to every thread, but is generally slow except when accesses are sequential or "coalesced." In addition to their typical out-of-order execution model, modern CPUs also support SIMD or vectorized instructions. The restrictions on the GPU imply that an effective algorithm on the GPU translates well to other shared memory architectures.

In this chapter, we present a parallelization for interpolation and a series of novel parallel schemes for spreading. The success of these algorithms relies on dividing these operations into trivially parallelizable tasks and parallel primitives. For the remainder of the chapter, we will treat the force densities and fluid velocity as the result of a black-box computation and focus primarily on the interpolation and spreading operations. Furthermore, we consider the simplified problem of evaluating the discrete counterparts of

$$E(\boldsymbol{X}) = \int_\Omega \delta(\boldsymbol{x} - \boldsymbol{X}) e(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \quad \text{and} \tag{4.1}$$

$$\ell(\boldsymbol{x}) = \int_\Gamma \delta(\boldsymbol{x} - \boldsymbol{X}) L(\boldsymbol{X}) \, \mathrm{d}\boldsymbol{X}, \tag{4.2}$$

where scalar-valued Eulerian function $e : \Omega \to \mathbb{R}$ is interpolated to Lagrangian point $\boldsymbol{X}$ and Lagrangian function $L : \Gamma \to \mathbb{R}$ is spread to Eulerian point $\boldsymbol{x}$. Because individual components of $\boldsymbol{u}$ and $\boldsymbol{f}$ are discretized at separate sets of points, we take the view that e and E are analogous to a single component of $\boldsymbol{u}$ and $\dot{\boldsymbol{X}}$, respectively, and ℓ and L to a

component of $\boldsymbol{f}$ and $\boldsymbol{F}$, respectively. The algorithms described below are then applied to each component of $\boldsymbol{u}$ or $\boldsymbol{F}$ individually. We use the notation Ω_h, n_ω, and g as defined in Section 2.1 for the grid underlying the Eulerian discretization. Likewise, we adopt the notation Γ_h to refer to a Lagrangian grid and n_γ its cardinality. Although we do not make the distinction while developing these algorithms, in the context of interpolation, Γ_h is the set of surface points corresponding to data sites, and in the context of spreading, it is the set of surface points corresponding to sample sites.

This chapter is organized into four sections. In Section 4.1, we discuss the serial and parallel interpolation and spreading algorithms. In Section 4.2, we demonstrate that the concurrency of the parallel algorithms scales with the number of points, independently of the Eulerian grid. We confirm the convergence of the method with these new algorithms. We illustrate the suitability of our algorithms to both GPUs and multicore CPU architectures with both weak and strong scaling tests. Finally, we summarize our findings in Section 4.3.

4.1 Algorithm design

Before tackling the tasks of interpolation and spreading, we require a few preliminary definitions. The discrete analog to the Dirac delta function, δ_h, is the tensor product of scaled, one-dimensional kernels, $\hat{\delta}_h(x) = h^{-1}\hat{\delta}_1(h^{-1}x)$. The kernel $\hat{\delta}_1$ is compactly supported. Let $\operatorname{supp}\psi$ denote the support of ψ and define

$$s[\hat{\delta}_1] = \max_{r\in[0,1)} |\operatorname{supp}\hat{\delta}_1(\cdot - r)\cap\mathbb{Z}| - 1 \tag{4.3}$$

to be the size of the support in terms of unit intervals. We do not change $\hat{\delta}_1$ during a simulation, so having chosen $\hat{\delta}_1$, we write $s = s[\hat{\delta}_1]$ for brevity. For any $\boldsymbol{X}\in\Omega\subset\mathbb{R}^d$, there are at most s^d Eulerian points, $\boldsymbol{x}\in\Omega_h$ for which $\delta_h(\boldsymbol{x}-\boldsymbol{X})$ is nonzero. We call these points the *support points* of $\delta_h(\cdot-\boldsymbol{X})$. We partition Ω into cells based on shared sets of support points. The points $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ belong to the same cell if the support points of $\delta_h(\cdot-\boldsymbol{X}_i)$ and $\delta_h(\cdot-\boldsymbol{X}_j)$ are equal. The cells are equilateral with side length h and contain one point from Ω_h, except, perhaps, for cells near the boundary. We extend those cells by adding ghost points to Ω_h so that every cell contains exactly one point in Ω_h or a ghost point. We call this expanded set $\bar{\Omega}_h$. We consider these cells to be the *de facto* grid cells. Figure 4.1 illustrates the grid cells and the effect of different choices of $\hat{\delta}_1$ has on their location. We also see that $\bar{\Omega}_h$ correspond to either grid cell centers or vertices. We will henceforth refer to them simply as the grid points. We can now identify any $\boldsymbol{X}\in\Omega$ with a grid point $\boldsymbol{x}\in\bar{\Omega}_h$ if they belong to the same grid cell, and since $\boldsymbol{x} = h(\boldsymbol{i}+\boldsymbol{g})$, we identify the cell by the integers $\boldsymbol{i}$, which are

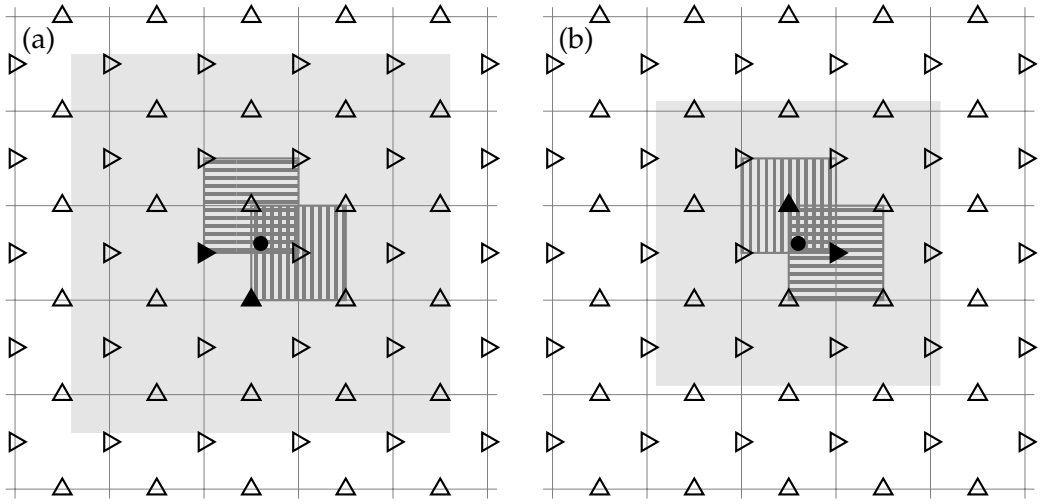


Figure 4.1: A region of a 2-dimensional domain, containing point $\boldsymbol{X}$, indicated by a black circle. Light gray lines indicate physical units in increments of h, and $\boldsymbol{X}$ has the same coordinates in each subfigure. Light gray boxes indicate the support of $\delta_h(\cdot - \boldsymbol{X})$. Subfigure (a) has $s = 4$ support points in each dimension, and (b) has $s = 3$. Horizontal vector components have staggering $\boldsymbol{g} = (0,\, 0.5)$ and are marked by right-pointing triangles, whereas upright triangles mark vertical vector components, which have staggering $\boldsymbol{g} = (0.5,\, 0)$. Those grid points within the gray boxes are also support points of $\boldsymbol{X}$. The horizontally- and vertically-striped gray boxes denote the *de facto* grid cell containing $\boldsymbol{X}$ for the horizontal or vertical component, respectively. The filled triangles mark the point $\boldsymbol{x} = h(\lfloor \boldsymbol{X} \rceil + \boldsymbol{g})$. The positions of the grid cells in (a) are typical of all even s, and the positions of the grid cells in (b) are typical of all odd s.

unique to the cell. We adopt the notation $\lfloor \boldsymbol{X} \rceil = \boldsymbol{i}$ for the function that maps the point $\boldsymbol{X}$ to the vector of integers identifying the grid cell containing $\boldsymbol{X}$.

We now turn our attention to the evaluation of $\delta_h(\boldsymbol{x} - \boldsymbol{X})$. We assume $\boldsymbol{x} \in \Omega_h$ and write

$$\begin{aligned} \delta_h(\boldsymbol{x} - \boldsymbol{X}) &= \delta_h(\boldsymbol{x} - h(\lfloor \boldsymbol{X} \rceil + \boldsymbol{g}) + h(\lfloor \boldsymbol{X} \rceil + \boldsymbol{g}) - \boldsymbol{X}) \\ &\equiv \delta_h(h\boldsymbol{\sigma} - \Delta\boldsymbol{x}) \\ &= \prod_{a=1}^{d} h^{-1} \hat{\delta}_1((\boldsymbol{\sigma} - h^{-1}\Delta\boldsymbol{x}) \cdot \boldsymbol{e}_a), \end{aligned} \tag{4.4}$$

where $\Delta\boldsymbol{x} = \boldsymbol{X} - h(\lfloor \boldsymbol{X} \rceil + \boldsymbol{g})$ is the displacement of $\boldsymbol{X}$ from its associated grid point, and $\boldsymbol{\sigma} = \lfloor \boldsymbol{x} \rceil - \lfloor \boldsymbol{X} \rceil$. We refer to $\boldsymbol{\sigma}$ as a *shift*. Shifts that result in a possibly nonzero value of δ_h are known *a priori* based on the kernel $\hat{\delta}_1$, and usually range from $-\lfloor s/2 \rfloor$ to $\lfloor (s-1)/2 \rfloor$ in each component. We can therefore assign an order to the shifts, $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_{s^d}$. As an example, Algorithm 4.1 computes $\boldsymbol{\sigma}_j$ in lexicographical order given s and d. Our implementation performs this step at compile time. We need only compute $\Delta\boldsymbol{x}$ once to be able to compute all nonzero values of $\delta_h(\boldsymbol{x} - \boldsymbol{X})$.

We need one more ingredient to construct $\mathcal{S}$. We let $\boldsymbol{x}_k$ be the be the k^{th} point in Ω_h. The

Algorithm 4.1 Shift construction

procedure SHIFT(j, s, d)
  ▷ **require**: $1 \leq j \leq s^d$
  ▷ **generate**: Shift σ_j
  $\sigma \leftarrow \mathbf{0}$ ▷ d zeros
  for $i = 1, \ldots, d$ **do**
    $\sigma_i \leftarrow \mathrm{mod}(j-1, s) - \lfloor s/2 \rfloor$
    $j \leftarrow \lfloor (j-1)/s \rfloor + 1$
  end for
  return σ
end procedure

procedure SHIFTS(s, d)
  ▷ **generate**: Shifts $\sigma_1, \ldots, \sigma_{s^d}$
  for $j = 1, \ldots, s^d$ **do**
    $\sigma_j \leftarrow$ SHIFT(j, s, d)
  end for
  return $\sigma_1, \ldots, \sigma_{s^d}$
end procedure

ordering of points dictates how we represent discrete values: we discretize the Eulerian function $e(\boldsymbol{x})$ as the vector $\boldsymbol{e}$, distinct from the basis vector $\boldsymbol{e}_a$, with k^{th} entry $e_k = e(\boldsymbol{x}_k)$. Define the *grid indexing function*

$$\#(\boldsymbol{i}) = \begin{cases} k, & \lfloor \boldsymbol{x}_k \rceil = \boldsymbol{i},\ \boldsymbol{x}_k \in \Omega_h \\ \varnothing, & \text{otherwise.} \end{cases} \tag{4.5}$$

The value $\varnothing$ indicates an Eulerian point outside or on the boundary of Ω, does not have a corresponding row in $\mathcal{S}$, and therefore does not contribute to spreading or interpolation. For example, each point in $\bar{\Omega}_h \setminus \Omega_h$ is indexed by $\varnothing$. We are now ready to construct $\mathcal{S}$. Let $\boldsymbol{X}_j$ be a Lagrangian point. The j^{th} column of $\mathcal{S}$ is zero except for up to s^d values where, for $i = 1, \ldots, s^d$, if $\#(\lfloor \boldsymbol{X}_j \rceil + \boldsymbol{\sigma}_i) = k \neq \varnothing$,

$$\mathcal{S}_{kj} = \delta_h \left(h \left(\boldsymbol{\sigma}_i + \lfloor \boldsymbol{X}_j \rceil + \boldsymbol{g} \right) - \boldsymbol{X}_j \right). \tag{4.6}$$

From a practicality standpoint, it is unnecessary to explicitly construct $\mathcal{S}$. We illustrate this with the serial spreading algorithm.

4.1.1 Serial spreading

Algorithm 4.2 lists an example serial implementation of spreading in pseudocode. The overall structure consists of two loops: a loop over the (indices of) points, and a loop over the (indices of) shifts. From this, we see that for a fixed choice of $\hat{\delta}_1$, and therefore s, the amount

Algorithm 4.2 Serial spreading

```
1:  procedure SEQ-SPREAD(Γ_h, Ω_h, L)
2:      ▷ generate: Approximate values of ℓ at each point in Ω_h
3:      σ_1, ..., σ_{s^d} = SHIFTS(s, d)                    ▷ Algorithm 4.1
4:      for i = 1, ..., n_γ do                              ▷ Loop over Lagrangian points
5:          x ← h(⌊X_i⌉ + g)                                ▷ X_i ∈ Γ_h, x ∈ Ω_h
6:          Δx ← x − X_i
7:          for j = 1, ..., s^d do                          ▷ Loop over shifts
8:              w ← δ_h(Δx + hσ_j)
9:              k ← #(⌊x⌉ + σ_j)
10:             if k ≠ ∅ then
11:                 ℓ_k ← ℓ_k + w · L_i
12:             end if
13:         end for
14:     end for
15:     return ℓ
16: end procedure
```

of work is $\mathcal{O}(n_\gamma)$, *i.e.*, independent of the Eulerian grid. The spread values are accumulated into a vector $\boldsymbol{\ell}$ (line 11). The target index, k (line 9), is computed using the grid indexing function, introduced in the previous section. Note that k depends on $\boldsymbol{\sigma}_j$ and $\boldsymbol{x}$, which in turn depends on $\boldsymbol{X}_i$, as seen on line 5. This means that k depends on both loop indices. There is no guarantee that unique pairs of the loop variables i and j will yield unique target indices. As a result, simply parallelizing one or both of the loops may lead to write contentions.

The property that runtime be independent of the Eulerian grid is desirable, as the number of Lagrangian points is often much fewer than the number of Eulerian grid points. In other words, many grid cells will be devoid of Lagrangian points, and unless nearby grid cells have Lagrangian points, there is no useful work to be done for that grid cell. An algorithm that depends on the Eulerian grid will invariably involve wasted computational resources. As we develop our parallel algorithms, we aim to preserve the grid-independence property. For example, this is achieved straightforwardly for interpolation, as we see in the next section.

4.1.2 Parallelization of interpolation

While the spreading matrix, $\mathcal{S}$, has approximately s^d nonzero entries per column, the interpolation matrix, $\mathcal{I}$, has approximately s^d nonzero entries per row, with one row per Lagrangian point. This property is beneficial for parallelization. The s^d potentially nonzero values in each row correspond to the shifts that produce a nonzero value of δ_h. Row i

interpolates to Lagrangian point $\boldsymbol{X}_i$. The grid point associated with $\boldsymbol{X}_i$ is at $\boldsymbol{x} = h(\lfloor \boldsymbol{X}_i \rceil + \boldsymbol{g})$ and $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{X}_i$. Since the shifts $\{\boldsymbol{\sigma}_j\}$ are known beforehand, we can compute $\delta_h(h\boldsymbol{\sigma}_j + \Delta\boldsymbol{x})$ and accumulate products independently of other Lagrangian points,

$$E_i = \sum_{\substack{j=1 \\ \#(\boldsymbol{\sigma}_j + \lfloor \boldsymbol{x} \rceil) \neq \varnothing}}^{s^d} \delta_h(h\boldsymbol{\sigma}_j + \Delta\boldsymbol{x}) e_{\#(\boldsymbol{\sigma}_j + \lfloor \boldsymbol{x} \rceil)},$$

where $\#(\boldsymbol{i})$ is defined in Equation (4.5). This is done for each Lagrangian point, for a total work proportional to n_γ.

Assigning one thread per Lagrangian point (*i.e.*, one thread per row of $\mathcal{I}$), this calculation can be performed in parallel, and since the i^{th} thread writes to E_i, there are no write contentions. This can be seen on lines 12 and 15 of Algorithm 4.3, where the accumulation happens in a temporary variable which is ultimately written to $\boldsymbol{E} = (E_i)$. In this case, the target index depends only on the loop variable i, and we can safely parallelize this loop. Because the number of products is approximately the same for each row, each thread does approximately the same amount of work. On architectures that enforce thread synchrony, such as GPUs, this means that we do not incur a penalty from threads waiting for other threads to finish.

Since each thread computes the appropriate δ_h-weights for its own row, it is unnecessary to construct $\mathcal{I}$ explicitly. Except allocating memory for $\boldsymbol{E}$, all of the work for this algorithm

Algorithm 4.3 Parallel interpolation

```
1:  procedure PAR-INTERPOLATE(Γ_h, Ω_h, e)
2:      ▷ generate: Approximate values of E at each point in Γ_h
3:      σ_1, ..., σ_{s^d} = SHIFTS(s, d)                    ▷ Algorithm 4.1
4:      for i = 1, ..., n_γ parallel do
5:          x ← h(⌊X_i⌉ + g)                                ▷ X_i ∈ Γ_h, x ∈ Ω_h
6:          Δx ← x − X_i
7:          v ← 0                                           ▷ Accumulator
8:          for j = 1, ..., s^d do
9:              w ← δ_h(Δx + hσ_j)
10:             k ← #(⌊x⌉ + σ_j)
11:             if k ≠ ∅ then
12:                 v ← v + w · e_k
13:             end if
14:         end for
15:         E_i ← v
16:     end for
17:     return E
18: end procedure
```

is parallel, so we expect to see near-perfect scaling. Additionally, other than using the grid spacing h for scaling in various places, the evaluation of δ_h, and information about boundaries, there is no dependence on the Eulerian grid. We now show how these properties can be maintained for the spreading operation.

4.1.3 Parallel spreading

We return now to spreading. Consider a fixed j in Algorithm 4.2 so σ_j is fixed. If every Lagrangian point inhabits its own grid cell, the support point for each Lagrangian point corresponding to σ_j is unique. In this case, we can spread to those support points without concern for write contentions. This is unlikely to occur in practice. On the other hand, if every Lagrangian point were in the same grid cell, values could be accumulated in parallel before being spread. This too is unlikely to occur in practice. We can instead employ the *segmented reduce* algorithm, which, given a list of values and a corresponding list of keys, will sum (reduce) consecutive values as long as their keys match. The result is a potentially shorter list of reduced values and a list of nonrepeating keys, though they may not be unique within the list. Suppose we were able to order the keys and values so that repeated keys are listed consecutively. The result of reducing this sorted data is a list of unique keys and a list where all values with the same key are combined. Assign each grid cell a unique index, and for each Lagrangian point, use the index for the grid cell it inhabits as its key. Then, segmented reduce accumulates values spread from Lagrangian points in the same grid cell for the given shift. Now, we have at most one value per grid cell and can write without contentions.

To ensure that repeated keys are listed consecutively, we use *key-value sort*, which, given a list of values and a corresponding list of keys, sorts values according to a partial ordering imposed on the keys. The result is a sorted list of keys and a permuted list of values, but key-value pairs remain unchanged. Computing keys and values, sorting by key, and applying the segmented reduce algorithm once per shift spreads all values. Because the keys are independent of the shift, sorting once per shift results in the same sorted list of keys each time. Instead of computing values and then sorting, we can first construct a permutation by sorting the indices of the Lagrangian points. This need only be done once per spreading operation. Then, we can use the permutation to compute values in sorted order for each shift. Now, we apply segmented reduction to the sorted list of keys and the newly constructed list of values. Finally, we write the reduced values to the output. Computing values, reducing, and writing for each shift completes the calculation.

Lastly, we need a suitable way to generate keys. A function κ that generates keys should be 1-to-1 with grid cells, or to points in $\bar{\Omega}_h$. For this reason, it is useful to formulate κ as a function of $\mathbb{Z}^d$ so that $\kappa(\lfloor \boldsymbol{X} \rceil)$ gives the key for Lagrangian point $\boldsymbol{X}$. The requirement that κ be injective will, in general, invalidate the grid indexing function as an otherwise good choice, as it maps each ghost point in $\bar{\Omega}_h$ to $\varnothing$. We can, however, use the key to compute grid indices: the point in Ω_h with key k also has grid index $\#(\kappa^{-1}(k))$. Because κ is injective, $\kappa^{-1}(k)$ is well-defined, and for shift $\boldsymbol{\sigma}$, $\#(\kappa^{-1}(k) + \boldsymbol{\sigma})$ yields the appropriate target index for writing to the output vector. Putting these pieces together, we have a complete parallel spreading algorithm, listed in Algorithm 4.4 and illustrated in Figure 4.2.

Lines 4–8 of Algorithm 4.4 construct the permutation by constructing a list of keys and a list of the indices of Lagrangian points, 1, 2, ..., n_γ, and sorting the indices according to

Algorithm 4.4 Parallel spreading

```
1:  procedure PAR-SPREAD(Γ_h, Ω_h, L)
2:      ▷ generate: Approximate values of ℓ at each point in Ω_h
3:      σ_1, ..., σ_{s^d} = SHIFTS(s, d)                          ▷ Algorithm 4.1
4:      for i = 1, ..., n_γ parallel do                           ▷ Loop over Lagrangian points
5:          K_i ← κ(⌊X_i⌉)                                        ▷ Sort key
6:          P_i ← i                                               ▷ Initial ordering
7:      end for
8:      sort {P_i} by {K_i}                                       ▷ i → P_i is a permutation
9:      q ← count unique {K_i}
10:     for j = 1, ..., s^d do                                    ▷ Loop over shifts
11:         {K'_i} ← {K_i}
12:         for i = 1, ..., n_γ parallel do                       ▷ Loop over Lagrangian points
13:             p ← P_i
14:             x ← h(⌊X_p⌉ + g)                                  ▷ X_p ∈ Γ_h, x ∈ Ω_h
15:             Δx ← x − X_i
16:             w ← δ_h(Δx + hσ_j)
17:             V_i ← w · L_p
18:         end for
19:         reduce {V_i} by {K'_i}                                ▷ Segmented reduce
20:         for i = 1, ..., q parallel do                         ▷ Loop over inhabited grid cells, q ≤ n_γ
21:             x ← h(κ^{-1}(K'_i) + g)
22:             m ← #(⌊x⌉ + σ_j)                                  ▷ ⌊x⌉ ≡ κ^{-1}(K'_i)
23:             if m ≠ ∅ then
24:                 ℓ_m ← ℓ_m + V_i
25:             end if
26:         end for
27:     end for
28:     return ℓ
29: end procedure
```

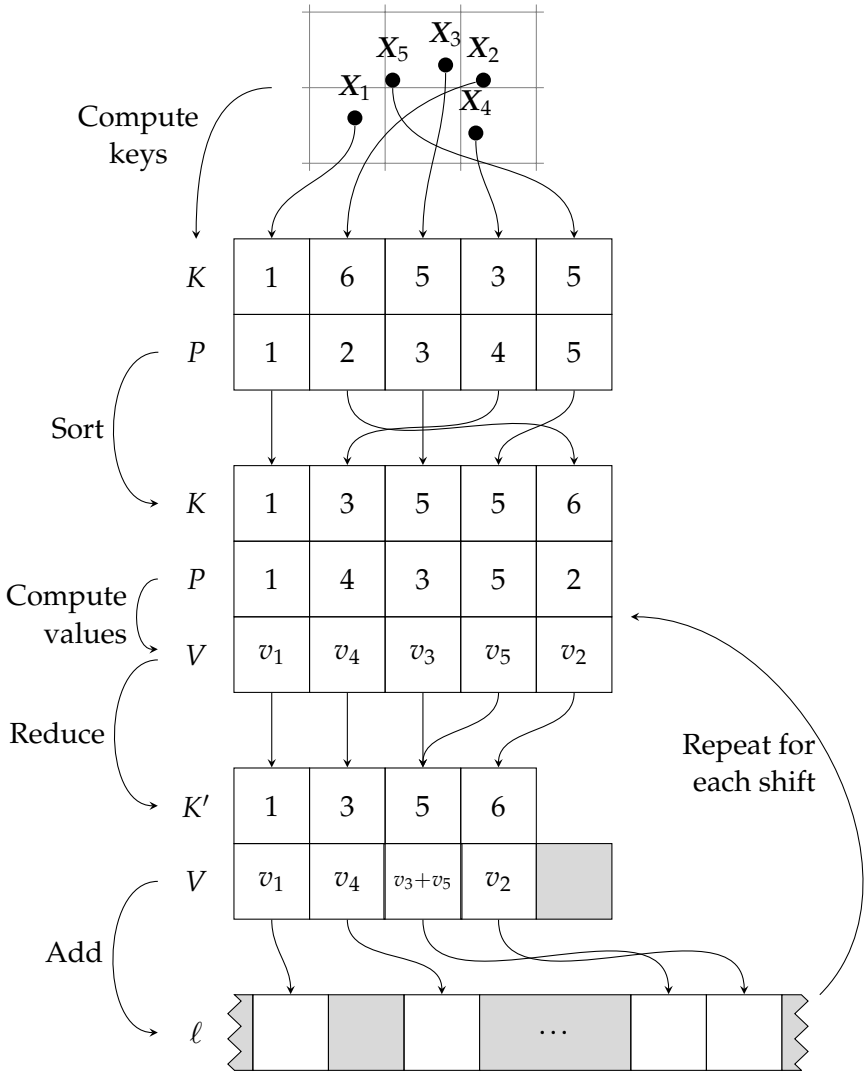


Figure 4.2: Schematic of Algorithm 4.4 for Lagrangian points $\boldsymbol{X}_1$ through $\boldsymbol{X}_5$ in a small region of Ω. Grey grid lines indicate the boundaries of grid cells and black-outlined boxes represent locations in memory. Greyed-out regions are unused memory. The labels K, P, K', V, ℓ, are the key, permutation, (reduced) value, reduced key, and output arrays, respectively. For this illustration, keys are computed sequentially, from left to right and bottom to top. Grid indices are ordered from left to right and assume that the grid continues horizontally, indicated by the ellipsis in the output array.

the keys. The i^{th} entry in the permuted list of indices gives the index of the i^{th} Lagrangian point in sorted order. On line 9, we define q to be the number of unique keys, which is also the number of reduced values to write. Since the list of sorted keys does not change for different shifts, we compute it once and reuse the value; see lines 20–26. Values, one per Lagrangian point, are computed and stored in a list $\boldsymbol{V}$ on lines 12–18. These values are then inputted to segmented reduce, line 19.

Algorithm 4.4 does not explicitly depend on the Eulerian grid, with some caveats dependent upon implementation details. Our implementation relies on the `thrust` library to provide the key-value sort, segmented reduce, and unique counting routines. Sorting is implemented as radix sort, which has a runtime of $\mathcal{O}(wn_\gamma/p)$, where p is the number of processors/threads, and w is the number of bits required to represent every key. In general, $w \propto \log_2 n$, for n elements to be sorted, but we use 32-bit integers for keys, so $w = 32$. It is reasonable to assume that this is true for most use cases, as there are approximately n_ω possible keys, and implementations of `BLAS` and `LAPACK` typically use 32-bit integers for indexing. However, extremely fine grids with more than 2^{32} grid points will require a larger data type to represent each key uniquely. In that case, w increases with a finer grid. Segmented reduction has a much more complicated relationship with the Eulerian grid. Parallelized segmented reduction has a worst-case runtime of $\mathcal{O}(n_\gamma/p)$, but the constant of proportionality depends on the density of points within inhabited grid cells. On the one hand, if all Lagrangian points inhabit the same grid cell, segmented reduce proceeds as a regular reduce, which is very fast. On the other hand, if every Lagrangian point inhabits its own grid cell, there is no work to be done, and any time spent by the algorithm is for naught. The density of Lagrangian points also affects the value of q but is bounded above by n_γ. For up to 2^{32} grid points, we expect the overall runtime of the spreading algorithm to be $\mathcal{O}(n_\gamma/p)$.

Finally, we remark that Algorithm 4.4 must synchronize threads once per shift. For choices of $\hat{\delta}_1$ where s^d is large, this can hurt performance. This requirement is written as a parallel loop (line 12) within a serial loop (line 10). To reduce the number of synchronizations, we must be able to compute and store several values at once. However, attempting to write multiple values at once may lead to contentions. In the next section, we develop algorithms that require fewer synchronizations by using more memory to spread several values at once while also avoiding write contentions.

4.1.4 Buffered spreading variants

Here we introduce a buffer in which to store incomplete calculations. It can be thought of as a set of temporary output vectors, which we will combine at the end of the algorithm to finish the calculation. These vectors have n_ω entries, so adding two of them requires as many operations, but is quite easily parallelized, and is provided by any parallel `BLAS` implementation. In this regard, these buffered variants depend explicitly on the Eulerian grid. The only additional requirement is that there is enough memory to hold the buffer.

To develop these variants, we note that the sorted list of keys and the permutation are the same for all shifts. Since the permutation dictates the order of the values to spread, and the list of keys decides the behavior of the segmented reduce, these steps are nearly identical for each shift, save for the effect each shift might have on the values. We can therefore compute the value of δ_h for several shifts at once, and use the same machinery to accumulate the values as if they were vectors. Though we do not implement it, we note that for choices of $\hat{\delta}_1$ that satisfy an even-odd condition [59], cleverly selecting which shifts are processed together can reduce the total number of calls to $\hat{\delta}_1$. Then, to avoid write contentions, each of the entries of the reduced vectors is written to a separate output vector of the buffer. We choose $\texttt{sz} \approx 10$ values to compute each iteration. Summing the vectors in the buffer yields the desired result.

Algorithm 4.5 lists the general form of the algorithm as pseudocode, and Algorithm 4.6 gives a minor modification. The difference between these two variants is the lifetime of the buffer: Algorithm 4.5 does not manage the buffer itself, but Algorithm 4.6 allocates and frees the buffer memory, limiting the lifetime of the buffer to the duration of the algorithm. The latter considers the memory allocation as part of the algorithm. The additional overhead from buffer allocation means that we never expect Algorithm 4.6 to outperform Algorithm 4.5; it is provided as an alternative for systems where maintaining a large, long-lived buffer may exhaust memory. We compare these algorithms and Algorithm 4.4 to Algorithm 4.2, and the efficacy of Algorithm 4.3 below.

4.2 Numerical results

Here we describe two types of test: unstructured Lagrangian points, in which points are placed randomly in the domain and generate a force independently from the other Lagrangian points, and structured Lagrangian points, in which the points comprise an elastic structure and forces are generated based on the configuration of the points as a whole. For these tests, we use a $16\,\mu\text{m} \times 16\,\mu\text{m} \times 16\,\mu\text{m}$ triply periodic domain with an initially

Algorithm 4.5 Preallocated buffer parallel spreading

procedure PA-PAR-SPREAD(Γ_h, Ω_h, $\boldsymbol{L}, \boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_{\mathtt{sz}}$)
  ▷ **require**: $\mathtt{sz} \geq 1$
  ▷ **generate**: Approximate values of ℓ at each point in Ω_h
  $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_{s^d} = \textsc{shifts}(s, d)$ ▷ Algorithm 4.1
  for $i = 1, \ldots, n_\gamma$ **parallel do** ▷ Loop over Lagrangian points
    $K_i \leftarrow \kappa(\lfloor \boldsymbol{X}_i \rceil)$ ▷ Sort key
    $P_i \leftarrow i$ ▷ Initial ordering
  end for
  sort $\{P_i\}$ **by** $\{K_i\}$ ▷ $i \to P_i$ is a permutation
  $q \leftarrow$ **count unique** $\{K_i\}$
  for $j = 0, \ldots, \lceil s^d / \mathtt{sz} \rceil - 1$ **do** ▷ Loop over shifts
    $\{K'_i\} \leftarrow \{K_i\}$
    for $i = 1, \ldots, n_\gamma$ **parallel do** ▷ Loop over Lagrangian points
      $p \leftarrow P_i$
      $\boldsymbol{x} \leftarrow h(\lfloor \boldsymbol{X} \rceil + \boldsymbol{g})$ ▷ $\boldsymbol{X}_p \in \Gamma_h$, $\boldsymbol{x} \in \Omega_h$
      $\Delta \boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{X}_i$
      for $k = 1, \ldots,$ **min**$(\mathtt{sz}, s^d - \mathtt{sz} \cdot j)$ **do**
        $w \leftarrow \delta_h(\Delta \boldsymbol{x} + h \boldsymbol{\sigma}_{\mathtt{sz} \cdot j + k})$
        $V_{ik} \leftarrow w \cdot L(\boldsymbol{X}_p)$ ▷ $V \in \mathbb{R}^{n_\gamma \times \mathtt{sz}}$
      end for
    end for
    reduce $\{V_{i\cdot}\}$ **by** $\{K'_i\}$
    for $i = 1, \ldots, q$ **parallel do** ▷ Loop over inhabited grid cells, $q \leq n_\gamma$
      $\boldsymbol{x} \leftarrow h(\kappa^{-1}(K'_i) + \boldsymbol{g})$
      for $k = 1, \ldots,$ **min**$(\mathtt{sz}, s^d - \mathtt{sz} \cdot j)$ **do**
        $m \leftarrow \#(\lfloor \boldsymbol{x} \rceil + \boldsymbol{\sigma}_{\mathtt{sz} \cdot j + k})$
        if $m \neq \varnothing$ **then**
          $\ell_{mk} \leftarrow \ell_{mk} + V_{ik}$
        end if
      end for
    end for
  end for
  return $\sum_{k=1}^{\mathtt{sz}} \boldsymbol{\ell}_k$
end procedure

Algorithm 4.6 On-the-fly buffer parallel spreading

procedure OTF-PAR-SPREAD(Γ_h, Ω_h, $\boldsymbol{L}$)
  ▷ **require**: $\mathtt{sz} \geq 1$
  ▷ **generate**: Approximate values of ℓ at each point in Ω_h
  for $i = 1, \ldots, \mathtt{sz}$ **do**
    $\boldsymbol{\ell}_k \leftarrow \boldsymbol{0}$
  end for
  return PA-PAR-SPREAD(Γ_h, Ω_h, $\boldsymbol{L}$, $\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_{\mathtt{sz}}$) ▷ Algorithm 4.5
end procedure ▷ Lifetime of $\boldsymbol{\ell}_k$ ends here

shear-like flow, $\boldsymbol{u} = (0,\, 0,\, \dot{\gamma}(y - 8\,\mu\text{m}))$, with shear rate $\dot{\gamma}$. Tests use a shear rate of 1000s^{-1} unless otherwise noted. This flow has a sharp transition at the periodic boundary $y = 0\,\mu\text{m}$, so a background force is added to maintain this transition and so that the initial flow is also the steady flow in the absence of other forces.

Serial and multicore CPU tests were performed on a single node with 48 Intel® Xeon® CPU E5-2697 v2 2.70GHz processors and 256 GB of RAM running CentOS Linux release 7.7.1908 (x86_64). Parallel CPU implementations use Intel®'s OpenMP library, `libiomp5` [53]. GPU tests used the same node with an NVIDIA® Tesla® K80 (2×GK210 GPU with 13 823.5MHz multiprocessors and 12 GB of global memory each). Only one of the GK210 GPUs was used. The CPU code was written in C++17 and the GPU code was written in C++/CUDA and used version 9.2 CUDA libraries [57]. Both the CPU and GPU codes were compiled using `clang` version 7.0.1. All tests are performed in double precision. We begin with tests using unstructured points, for which both of these architectures were used.

4.2.1 Unstructured Lagrangian points

Consider a set of n Lagrangian points randomly placed in the domain described above. We imagine the Lagrangian points to be tethered to their initial positions. The fluid solver is not invoked for these tests. This allows us to perform tests with large n without also waiting for the fluid solver. Instead, at each timestep, we interpolate the fluid velocity to each of the Lagrangian points and predict their updated positions. Using these predicted positions, we compute a Hookean force for each Lagrangian point with spring constant $k = 1 \times 10^{-2}\,\text{dyn/cm}$. We spread these forces from the predicted positions to the fluid grid, but do not use them to update the fluid velocity. This ensures that the points do not settle into a steady position so the spreading and interpolation operations receive new data each time step. We again interpolate the velocity to the Lagrangian points and update the position of the Lagrangian points. While the interpolated velocities at the end of the timestep are the same as those computed at the beginning, this is done by analogy to the full IB solver, which interpolates fluid velocities twice and spreads forces once per timestep.

In the sections that follow, we use this test to compare the performance of the parallel algorithms to their serial counterparts and, for the spreading variants, to each other. We first consider the effects of background grid refinement.

4.2.1.1 Dependence on background grid

The serial Algorithms 4.2 and 4.3 do not explicitly depend on the size of the fluid grid. With perhaps the exception of the sorting and reducing steps, Algorithm 4.4 also does not depend on the size of the Eulerian grid. Algorithms 4.5 and 4.6, on the other hand, ultimately sum their buffer vectors, which have one entry per grid point. Algorithm 4.6 also incorporates the allocation of these buffers. Using a few different fluid grids, we investigate whether Algorithms 4.3 and 4.4 are independent of the grid in practice, and how the grid dependence affects the runtime of Algorithms 4.5 and 4.6.

Table 4.1 lists timing results for $n = 2^{16}$ Lagrangian points and 16, 32, 64, and 128 grid points per 16 µm. The rows with device listed as $1 \times$ CPU correspond to the serial algorithms and will serve as a reference point for the rest of the section. If the serial algorithms depend on the fluid grid, they do so only mildly. In fact, under close scrutiny, it seems that these deviations are due to hardware-level differences in the integer multiplications and additions used in computing sort keys and grid indices. We can therefore expect each of the algorithms to exhibit a mild variation in runtime for different grid refinements.

The speedup for these schemes is illustrated in Figure 4.3. Algorithm 4.3 is independent

Table 4.1: Average time per call for interpolating to and spreading from 2^{16} Lagrangian points from 1000 time steps on different devices and grid configurations. Grid refinement is the number of grid points per 16 µm in each dimension. Times are reported in seconds.

		Grid refinement			
Device	Algorithm	16	32	64	128
$1 \times$ CPU	4.2	1.33249	1.33621	1.33840	1.37281
	4.3	1.29633	1.31373	1.30763	1.35101
$16 \times$ CPU	4.3	0.09890	0.09928	0.09974	0.10624
	4.4	0.23282	0.26431	0.25783	0.26590
	4.5	0.12803	0.14213	0.15215	0.20107
	4.6	0.12965	0.14242	0.14766	0.21874
$1 \times$ GPU	4.3	0.01253	0.01317	0.01722	0.01816
	4.4	0.03930	0.04020	0.04215	0.04755
	4.5	0.01715	0.02049	0.02370	0.03656
	4.6	0.01804	0.02198	0.02873	0.07288

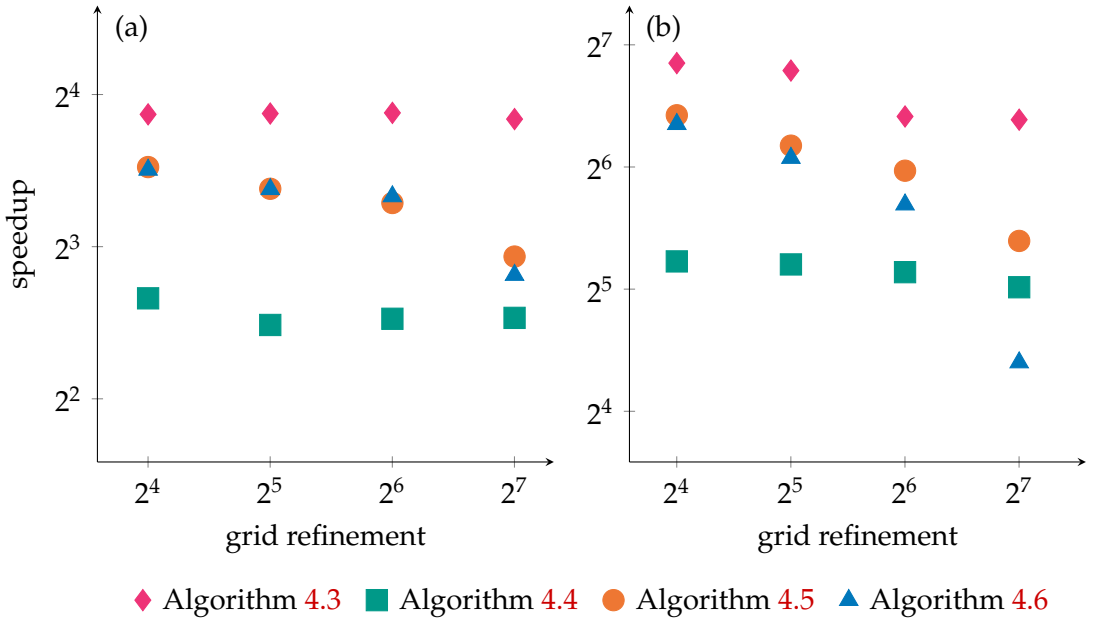


Figure 4.3: Speedup of Algorithm 4.3 (interpolation) and Algorithms 4.4–4.6 (spreading) compared to their serial counterparts on 2^{16} randomly placed Lagrangian points for different grid refinements on (a) 16 CPUs and (b) the GPU.

of the fluid grid, as expected. Any grid dependence introduced by the sort and reduce steps of Algorithm 4.4 is not obvious for the grids presented. On the other hand, the degradation of speedup for the sweep-fused Algorithms 4.5 and 4.6 is apparent for finer grids. For small problems with 128^3 or fewer grid points, one gets better performance from the sweep-fused variants, except for Algorithm 4.6 on the GPU. This can be attributed to the slower allocation of the buffer for a finer grid. For problems where the grid has 256^3 or more grid points, we expect Algorithm 4.4 to be the fastest choice for spreading. Because our fluid solver fits in GPU memory only for fewer than 128^3 grid points, for the remainder of this work we consider only a grid with a grid refinement of 64 ($h = 0.25\,\mu\text{m}$). We can imagine using this algorithm on a less capable device with more limited memory, so we restrict ourselves to Algorithm 4.6 for spreading, computing $\texttt{sz} = 8$ values per sweep, to minimize the lifetime of the buffer used in spreading while still enjoying the benefit of using the buffer.

4.2.1.2 Strong scaling

It is commonly the case that one wishes to employ parallelization to improve runtimes for a problem of interest. To illustrate this improvement, we now consider how runtime varies for the test problem with $n = 2^{16}$ Lagrangian points, a grid refinement of 64 (grid size $h = 0.25\,\mu\text{m}$), and Algorithm 4.6 with different numbers of threads. We use up to 32

threads on the CPU and 64–4096 threads on the GPU. For a fixed problem, we ideally wish to see the runtime using $2p$ threads to be half of that using p threads. In other words, using twice as many threads yield an ideal speedup of 2.

Figure 4.4 shows the results of these tests. Speedup is measured relative to the serial interpolation and spreading implementations. The trendlines estimate that increasing computing resources by a factor of two decreases runtime by a factor of about 1.91 for CPU and GPU interpolation and by a factor of about 1.85 for CPU spreading. It is not trivial to limit the number of threads used by `thrust` for work done on the GPU, so the key-value sort and segmented reduce use as many threads as `thrust` decides is prudent. While the trendline indicates a decrease in runtime by a factor of 1.91 as well, this is merely an approximation.

Parallel CPU interpolation using a single processor is identical to the serial CPU interpolation, so its plot in Figure 4.4(a) passes through $(1, 1)$. The same is not true of parallel spreading using a single processor compared to serial spreading. Because of the additional sort step in the parallel spreading algorithm, single-threaded Algorithm 4.4 is about 12% slower than its serial counterpart. The CPU code also enjoys the benefit of using vector

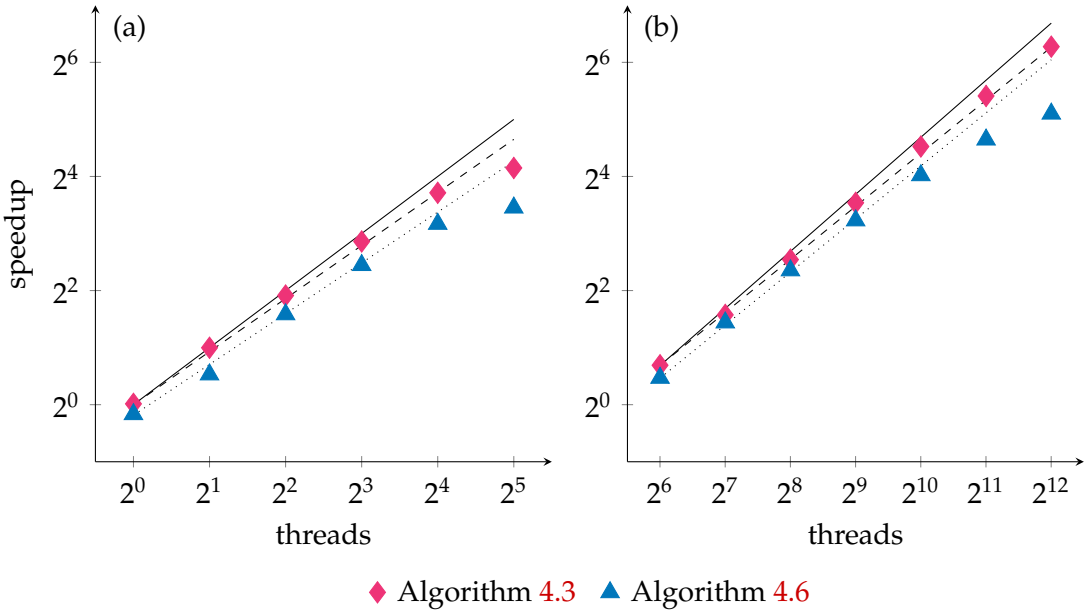


Figure 4.4: Strong scaling results for parallel interpolation and Algorithm 4.6 with grid spacing $h = 0.5\,\mu\text{m}$ (a grid refinement of 64) for 2^{16} randomly placed Lagrangian points in a $16\,\mu\text{m} \times 16\,\mu\text{m} \times 16\,\mu\text{m}$ triply periodic domain for (a) 1–32 CPU cores, and (b) 64–4096 threads on the GPU. Speedup is measured relative to serial Algorithms 4.3 and 4.2. The solid black lines show the trendline for ideal speedup. The dashed or dotted lines give the initial trend for interpolation and spreading, respectively.

registers for parts of the computation. The GPU requires 64 threads to match the speed of a single CPU core. Even at 4096 threads, interpolation on the GPU shows no indication of plateauing. The final data point for that curve shows a speedup of approximately 77×. Figure 4.3(b), on the other hand, shows that the maximum speedup we can expect for this problem is approximately 85×, which the trendline in Figure 4.4(b) predicts will occur at approximately 4480 threads. Thus, we can expect the plateau for interpolation on the GPU to be very abrupt. This indicates a hardware limitation, the likely culprit being exhaustion of register memory. The plateauing of the CPU curves is not a limitation of the algorithm for the CPU. Despite having 48 cores, the test using 32 cores did not utilize them at full capacity. Using fewer cores, on the other hand, was able to maintain full utilization for the duration of the test. If not for having a comparatively limited number of CPU cores, we expect to see the CPU trend continue.

Up to hardware limitations, it seems that the algorithm scales without bound on either the CPU or the GPU. Overall, trends for the CPU and GPU are very similar, with nearly perfect scaling. Because of these similarities and the greater capacity for speedup, we will restrict ourselves to the GPU for the remainder of the chapter, but expect any conclusions to hold for the CPU as well.

4.2.1.3 Weak scaling

In contrast, with improved computing resources, we may wish to solve bigger problems. The ideal parallel algorithm solves a problem with p threads in the same amount of time as it solves a twice bigger problem with $2p$ threads. Here, we place between 2^{16} and 2^{19} points randomly in a fixed domain with $h = 0.25\,\mu\text{m}$. As we increase the number of points, we increase the number of threads proportionally, between 128 and 1024.

Table 4.2 lists runtimes for increasing threads and problem size on the GPU. Interpolate scales nearly perfectly with a difference of 15 ms (~3%) increase between the problem with 128 threads and 2^{16} Lagrangian points and that with 1024 threads and 2^{19} points. Spread, on the other hand, decreases in time as the problem size increases. This speedup is artificial, and should not be expected in general. In the $n = 2^{16}$ case, there is 1 Lagrangian point for every 4 grid cells, on average. When $n = 2^{19}$, the density increases to 4 for every grid cell. As a result, it becomes increasingly unlikely to find a cell containing no Lagrangian points. This means that writing the values to the output vector(s) becomes increasingly coalesced, which, in turn, reduces the number of writes to global memory and vastly improves the speed of the write overall. Typical use of the IB method does not have Lagrangian points in

Table 4.2: Weak scaling results for interpolation and spreading for p threads and n randomly placed Lagrangian points in a 16 µm × 16 µm × 16 µm triply periodic domain with $h = 0.25$ µm on the GPU. The average time per call is reported in seconds. N is the number of samples taken.

p	n	interpolation $N = 20000$	spreading $N = 10000$
128	2^{16}	0.43930	0.47632
256	2^{17}	0.44918	0.46503
512	2^{18}	0.45072	0.44533
1024	2^{19}	0.45442	0.43561

every grid cell, but the recommendation that Lagrangian points on connected structures be spaced $0.5h$–h apart typically yields 1–8 Lagrangian points in each occupied grid cell, depending on the type of structure. To illustrate this, we consider a more typical use of the IB method in the upcoming section.

4.2.2 Elastic objects

We are motivated by the desire to simulate the motion of cells immersed in a fluid. Cells are not randomly generated points, but cohesive structures, kept together by elastic forces and the near-constant volume enclosed by their membranes. In this section, we replace the randomly placed points with points sampled on the surface of either a sphere or an RBC. Each test lists the number of data and sample sites used. Data and sample sites are generated using the method described by [60]. In this case, we invoke the fluid solver, so that as the object deforms, the force it imparts on the fluid will affect the fluid velocity. The sphere and RBC are elastic, obeying Skalak's Law [61], as described in Section 2.2. These tests require a time step of $\Delta t = 0.1$ µs for stability. With $\dot{\gamma} = 1000\,\mathrm{s}^{-1}$, a Lagrangian point requires at least 32 timesteps to transit a grid cell, so unlike the tests using randomly placed points above, there will be considerably more redundant computation. We first validate the fluid solver with the elastic sphere before performing scaling tests, similar to those above, with RBCs.

4.2.2.1 Convergence study

To test the convergence of the fluid solver and cell representation, consider an object that obeys Skalak's Law and is spherical at rest. Deform the object from its rest configuration by

stretching it by a factor of 1.1 in the z direction and compressing it by a factor of 1.1 in the y direction to maintain a fixed volume. For this test, $\dot{\gamma}$ is zero, so the fluid velocity is initially zero, and the object tends toward its rest configuration throughout the simulation. We allow the object to relax for 16 µs and compare errors generated by successive grid refinements of 16, 32, 64, and 128 points per 16 µm. For each grid, we use a fixed set of $n_d = 625$ data sites, sampled approximately uniformly on the surface of the sphere, and choose n_s so that sample sites are approximately uniform and roughly $h/1.1$ apart, so that sample sites are roughly h apart initially. For $h = 1$ µm, $n_s = 220$, and refinement by a factor of 2 increases n_s by a factor of 4, for a maximum number of 14080 sample sites for these tests. A thin interface that generates a force will cause a jump in the normal stress across the interface, which the IB method may not recover. Therefore, we anticipate first-order convergence for the fluid velocity and data site positions.

Tables 4.3 and 4.4 show the convergence of fluid velocity and data sites, respectively. To compute errors in the fluid velocity, we construct a cubic spline from the velocities on the finer grid and evaluate the spline at the grid points of the coarser grid. Each of the interfaces uses the same number of data sites, allowing us to directly compare their Cartesian coordinates between simulations. We recover approximately first-order convergence for both fluid velocity and data sites' Cartesian positions. Having established asymptotic convergence of our IB solver, we continue by performing scaling tests with RBCs.

4.2.2.2 Strong scaling

We again wish to see how these algorithms can help speed up the runtime of a fixed problem. Here, we consider tests with a single RBC and with 4 RBCs. To construct the RBCs, we now use $n_d = 864$ data sites, for an initial data site spacing of approximately $1.6h$, and $n_s = 8832$ sample sites per cell, for an initial sample site spacing of approximately $0.5h$. We

Table 4.3: Convergence of the fluid velocity for a deformed sphere returning to its rest configuration in a 16 µm × 16 µm × 16 µm triply periodic domain at $t = 160$ µs. The finest grid, with $h = 0.125$ µm uses time step $\Delta t = 0.025$ µs.

h (µm)	Δt (µs)	$\|\boldsymbol{u}_h - \boldsymbol{u}_{0.5h}\|_2$	order	$\|\boldsymbol{u}_h - \boldsymbol{u}_{0.5h}\|_\infty$	order
1.00	1.6	2.2274×10^{-2}		7.4187×10^{-2}	
0.50	0.4	4.3280×10^{-4}	5.71513	1.9083×10^{-3}	5.28079
0.25	0.1	1.4684×10^{-4}	1.55951	1.0847×10^{-3}	0.81497

Table 4.4: Convergence of data sites' Cartesian coordinates for a deformed sphere returning to its rest configuration in a 16 µm × 16 µm × 16 µm triply periodic domain at $t = 16$ µs. For each grid, we track 625 data sites on the sphere. The finest grid, with $h = 0.125$ µm used $n_s = 14080$ sample sites.

h (µm)	n_s	$\|\boldsymbol{X}_h - \boldsymbol{X}_{0.5h}\|_2$	order	$\|\boldsymbol{X}_h - \boldsymbol{X}_{0.5h}\|_\infty$	order
1.00	220	2.9611×10^{-3}		4.7812×10^{-3}	
0.50	880	7.2997×10^{-4}	2.02020	1.1687×10^{-3}	2.03253
0.25	3520	3.5909×10^{-4}	1.02351	6.0956×10^{-4}	0.93902

use a time step of $\Delta t = 0.1$ µs to simulate the motion of the cells for 1 ms.

Figure 4.5 shows the speedup observed with increasing threads for 1 and 4 RBCs for 64–4096 threads on the GPU. We again see that the initial speedup for interpolation is nearly linear with increased threads. In Figure 4.5(a), there is a sharp plateau that corresponds to every data site having its own thread. In other words, there are more threads than there is work to do since we track only 864 data sites for a single RBC. Figure 4.5(b), on the other hand, has 3456 data sites, so the trend continues for 512–4096 threads. In this case, we expect this graph to plateau beyond 4096, when each data site has its own thread. However, we do not expect the trend to continue with more cells, as the presumed maximum number of concurrent threads for interpolation is 4480, as discussed in Section 4.2.1.2. Comparing Figure 4.5(a) to Figure 4.5(b), we see that the speedup in spreading is also dependent on the amount of work. This indicates that, as with interpolation, the maximum speedup for spreading is limited by hardware, rather than being a limitation of the algorithm.

The trend lines for these tests indicate that increasing computing resources by a factor of two decreases runtime by a factor of about 1.87 for these algorithms. Again, this is merely an approximation as the sort and reduction steps of the spreading algorithm are provided by `thrust`, and therefore are not limited to the listed number of threads. The similarity between the result of the tests with RBCs and with randomly placed points indicates that the distribution of points does not have a marked impact on the efficacy of the parallelization for a fixed problem. We now see if the same holds for weak scaling tests.

4.2.2.3 Weak scaling

To see how the algorithms scale given more computing resources, we increase the number of cells in the domain and the number of threads proportionally. We construct each cell with $n_d = 832$ data sites and $n_s = 8832$ sample sites, as before. We place between 1 and 8 cells

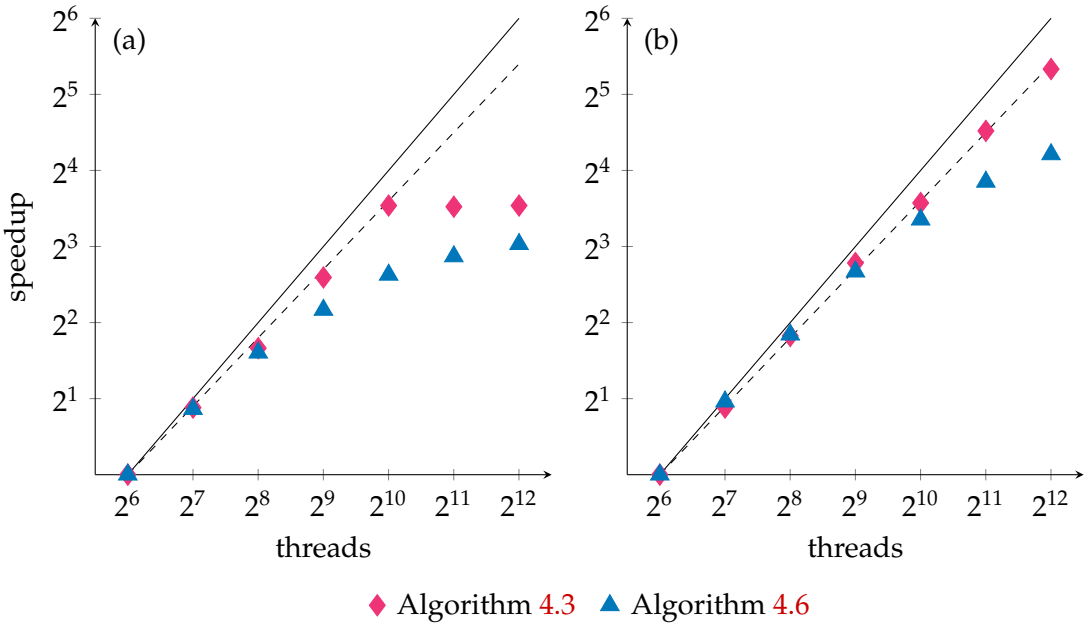


Figure 4.5: Speedup of Algorithms 4.3 and 4.6 with increasing numbers of threads compared to 64 threads on the GPU for (a) 1 and (b) 4 RBCs. Speedup is measured relative to the time taken for each algorithm using 64 threads on the GPU. Dashed lines indicate trends, and solid lines indicate ideal scaling.

in the domain, while threads increase from 64 to 512. Using a time step of $\Delta t = 0.1\,\mu s$, we simulate the motion of these cells for 1 ms.

In Section 4.2.1.3, we observe that, as a side effect of increasing point density per grid cell, runtime for spreading decreases as the number of points and threads increases. Here, the cells are initially far enough apart as to not have any overlapping support points. As a result, while individual grid cells may contain several Lagrangian points, average point density is still low, so we do not expect to see the same reduction in runtime as observed previously.

Table 4.5 shows the runtimes for increasing number of RBCs and threads. We observe the near-perfect scaling we saw with random Lagrangian points. For both interpolation and spreading, we see that the runtimes are nearly constant: the slowest and fastest times differ by less than 2 ms and 7 ms, for interpolation and spreading, respectively. After an initial drop in runtime between one RBC with 64 threads and two RBCs and 128 threads, runtimes even off and begin to increase, in contrast with the results in Section 4.2.1.3.

Table 4.5: Weak scaling results for interpolation and spreading for an increasing numbers of RBCs (cells column) and threads. Each RBC has $n_d = 864$ and $n_s = 8832$. The average time per call is reported in seconds. N is the number of samples taken.

p	cells	interpolation $N = 20000$	spreading $N = 10000$
64	1	0.01079	0.11881
128	2	0.01165	0.11219
256	4	0.01171	0.11214
512	8	0.01199	0.11354

4.3 Summary

We presented Algorithm 4.3 for interpolation in parallel and introduced a novel parallel algorithm, Algorithm 4.4, for spreading forces from time-dependent interfaces in the immersed boundary method. These algorithms have a runtime that is independent of the Eulerian grid. This makes the parallelization useful for cases where Lagrangian points inhabit only a small percentage of the Eulerian grid cells. We also introduced two variants for spreading which trade dependence on the Eulerian grid, in the form of a few vector additions and memory allocation, for improved runtimes on small enough grids.

These algorithms exhibit nearly ideal scaling, on both the CPU and GPU, for problems of fixed size and an increasing number of threads, as well as for problems of increasing size and number of threads. We observed that on the GPU, larger problem sizes led to higher speedup plateaus, indicating that larger problems on more capable hardware will achieve even larger speedups than presented here.

References

[46] P.-Y. Chuang, O. Mesnard, A. Krishnan, and L. A. Barba, *PetIBM: toolbox and applications of the immersed-boundary method on distributed-memory architectures*, Journal of Open Source Software, 3 (2018), p. 558.

[47] T. G. Fai, B. E. Griffith, Y. Mori, and C. S. Peskin, *Immersed boundary method for variable viscosity and variable density problems using fast constant-coefficient linear solvers I: numerical method and results*, SIAM Journal on Scientific Computing, 35 (2013), pp. B1132–B1161.

[48] B. E. Griffith, *An accurate and efficient method for the incompressible navier–stokes equations using the projection method as a preconditioner*, Journal of Computational Physics, 228 (2009), pp. 7565–7595.

[49] B. E. Griffith, *Immersed boundary model of aortic heart valve dynamics with physiological*

driving and loading conditions, International Journal for Numerical Methods in Biomedical Engineering, 28 (2011), pp. 317–345.

[50] B. E. Griffith, R. D. Hornung, D. M. McQueen, and C. S. Peskin, *An adaptive, formally second order accurate version of the immersed boundary method*, Journal of Computational Physics, 223 (2007), pp. 10–49.

[51] B. E. Griffith, R. D. Hornung, D. M. McQueen, and C. S. Peskin, *Parallel and adaptive simulation of cardiac fluid dynamics*, in Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications, M. Parashar and X. Li, eds., John Wiley & Sons, Inc., Hoboken, NJ, USA, 2010, pp. 105–130.

[52] B. E. Griffith and X. Luo, *Hybrid finite difference/finite element immersed boundary method*, International Journal for Numerical Methods in Biomedical Engineering, 33 (2017), p. e2888.

[53] Intel, *Intel oneAPI*, 2021, https://software.intel.com/content/www/us/en/develop/tools/oneapi.html.

[54] S. K. Layton, A. Krishnan, and L. A. Barba, *cuIBM – a GPU-accelerated immersed boundary method*, arXiv.org, (2011), https://arxiv.org/abs/1109.3524.

[55] D. M. McQueen and C. S. Peskin, *Shared-memory parallel vector implementation of the immersed boundary method for the computation of blood flow in the beating mammalian heart*, The Journal of Supercomputing, 11 (1997), pp. 213–236.

[56] O. Mesnard and L. A. Barba, *Reproducible and replicable computational fluid dynamics*, Computing in Science Engineering, 19 (2017), pp. 44–55.

[57] NVIDIA, P. Vingelmann, and F. H. Fitzek, *CUDA*, 2021, https://developer.nvidia.com/cuda-toolkit.

[58] S. Patel, *aeroCuda: the GPU-optimized immersed solid code*, Undergraduate thesis, Harvard, Harvard University, June 2012.

[59] C. S. Peskin, *The immersed boundary method*, Acta Numerica, 11 (2002), pp. 479–517.

[60] V. Shankar, R. M. Kirby, and A. L. Fogelson, *Robust node generation for meshfree discretizations on irregular domains and surfaces*, SIAM Journal on Scientific Computing, 40 (2018), pp. A2584–A2608.

[61] R. Skalak, A. Tozeren, R. P. Zarda, and S. Chien, *Strain energy function of red blood cell membranes*, Biophysical Journal, 13 (1973), pp. 245–264.

[62] P. Valero-Lara, F. D. Igual, M. Prieto-Matías, A. Pinelli, and J. Favier, *Accelerating fluid–solid simulations (lattice-boltzmann & immersed-boundary) on heterogeneous architectures*, Journal of Computational Science, 10 (2015), pp. 249–261.

CHAPTER 5

WHOLE BLOOD SIMULATIONS

As RBCs move towards the center of a blood vessel, they may encounter platelets on their way, but the relative size and deformability of the RBC means a platelet is ejected from the RBC's path, ultimately relegated to the RBC-free layer along the vessel wall. This is the margination process; it affects platelets and leukocytes (white blood cells) alike, and is the focus of many studies [65, 66, 68–70, 72, 75, 76, 78, 80, 85, 86, 89, 90]. Platelets are key players in vascular maintenance. From their marginated positions, platelets survey the vessel wall for injury. Injury sites release chemical signals which activate the platelet. This, in turn, leads the platelet to bind to the injury site and release its own chemical signals to recruit further platelets, which eventually results in the formation of a thrombus. All of this occurs in flowing blood, which sweeps these chemical signals downstream. While mechanisms for platelet activation have been proposed for low and pathologically high shear rates, the case of physiologically high shear rates is undecided [71]. We assume that platelets have marginated and focus on this shear regime.

Models of platelet motion over a thrombus indicate that there are stagnation zones immediately upstream and downstream of the thrombus, where the fluid velocity is very slow, even when the thrombus protrudes only a few microns from the vessel wall [84, 87]. Platelets that enter these regions spend a disproportionate amount of time there. The endothelial cell nucleus also protrudes into the vessel approximately 1 µm. Moreover, the endothelial bumps are roughly periodic. If the endothelium creates a stagnation zone, the trailing zone from one protrusion might lead into the leading zone of the subsequent protrusion. This may allow for the sequestration of platelets or chemical signals. However, typical models of platelet-wall interaction model the endothelium as a flat surface [85, 88].

In this chapter, we combine the work from the previous four chapters to validate the RBF-IB methodology for the simulation of whole blood. We use it to compare the flow of whole blood across bumpy and flat walls and characterize the behavior and interactions of platelets. This chapter is divided into 3 parts. In Section 5.1, we validate our methods and models. In Section 5.2, we present results from whole blood simulations. We summarize

our results and discuss their implications in Section 5.3.

5.1 Numerical verification

We have taken a few departures from the traditional IB method and RBC models. In this section, we verify that these changes do not alter the efficacy of the method or expected behaviors of the RBCs. We use a few different choices for $\hat{\delta}_1$ in this section; they are shown in Figure 5.1.

5.1.1 Convergence study

In this section, we perform a series of tests on a single perturbed RBC undergoing relaxation. We simplify the RBC model and use only Skalak's Law. We expect the IB method to approximate the fluid velocity at first order for thin shells, as it cannot recover the pressure jump across the interface. We stretch the RBC by a factor of 1.1 in the z direction and compress it in the x direction to maintain its reference volume. We place the cell in the center of a 16 µm × 16 µm × 16 µm domain with homogeneous Dirichlet boundary conditions in the y direction and periodic boundaries elsewhere. The fluid velocity is initially zero. The cell is then allowed to relax for 180 µs.

We discretize the RBC using the Bauer spiral [64],

$$\begin{aligned} \varphi_j &= \sin^{-1}(-1 + (2i-1)/N), \\ \theta_j &= \operatorname{mod}\left(\sqrt{N\pi}\varphi_j + \pi,\, 2\pi\right) - \pi, \end{aligned} \tag{5.1}$$

where N is either n_d or n_s, and $\operatorname{mod}(a,\, n) = a - |n|\,\lfloor a/|n| \rfloor$ is the positive modulo function. The Bauer spiral helpfully avoids the poles, where $\det(\mathcal{G}_0(\boldsymbol{Z}_{S^2})) = 0$, while maintaining a reasonable distribution of points on the surface of the RBC. We use the 3-point kernel $\hat{\delta}_1$

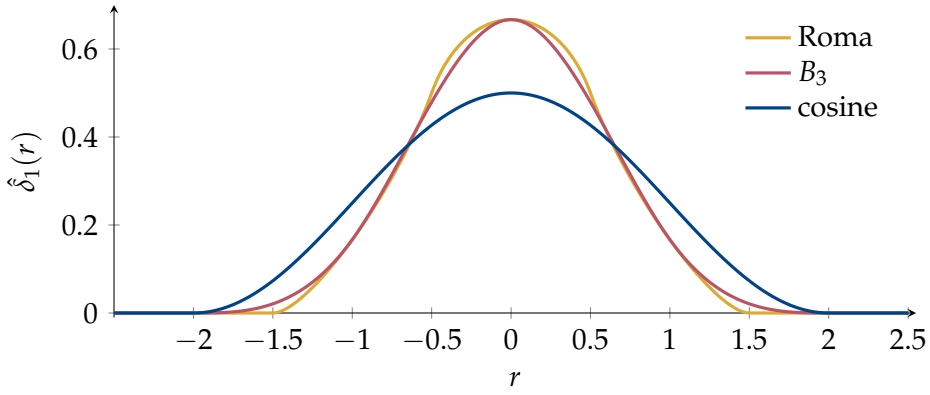


Figure 5.1: An illustration of the 3-point Roma [82], 4-point B_3 B-spline, and 4-point cosine [81] kernels used in this chapter.

derived by Roma *et al.* [82], shown in Figure 5.1, for spreading and interpolation and the 2-stage RK method described in Section 2.4 to advance the fluid velocity. Fluid grids are chosen to have $20r$ grid points per 16 µm in each direction for r from 1 to 6. On successive grids, we compare the fluid velocity at cell centers in a regular grid of 20^3 cells and surface positions at 1000 surface points. For convergence of order p, we expect the ratio of successive errors to satisfy

$$\frac{\epsilon_r}{\epsilon_{r+1}} = \left| \frac{((r+1)/r)^p - 1}{((r+2)/(r+1))^{-p} - 1} \right|,$$

which we solve numerically for p. We use the machinery of Sections 2.3.3 and 3.2.4.2 to approximate quadrature weights in material coordinates such that

$$\omega_j \det(\mathcal{G}_0(\boldsymbol{Z}_{S^2}))^{-1/2} = \omega_j / \cos(\varphi_j)$$

is analogous to $\mathrm{d}\theta\,\mathrm{d}\varphi$ at Lagrangian point $\boldsymbol{X}_j$. This enables us to compute the errors in $\boldsymbol{X}$ using discrete versions of the L_2 and L_∞ norms,

$$\|\boldsymbol{X}(\theta,\,\varphi)\|_2^2 = \int_{S^2} \boldsymbol{X}(\theta,\,\varphi)\cdot\boldsymbol{X}(\theta,\,\varphi)\,\mathrm{d}\theta\,\mathrm{d}\varphi \quad \text{and} \tag{5.2}$$

$$\|\boldsymbol{X}(\theta,\,\varphi)\|_\infty^2 = \max_{(\theta,\,\varphi)} \boldsymbol{X}(\theta,\,\varphi)\cdot\boldsymbol{X}(\theta,\,\varphi), \tag{5.3}$$

respectively.

Tables 5.1 and 5.2 show L_2 and L_∞ errors in $\boldsymbol{u}$ and $\boldsymbol{X}$ between successive grids. We observe first-order convergence, as expected. Satisfied with the convergence of our implementation, we henceforth continue using the Bauer spiral to discretize RBCs and use the complete RBC model, which we verify in the next section.

5.1.2 Tumbling and tank-treading

Few RBC models include viscoelastic forces. Fedosov *et al.* use a particle-based method to simulate the fluid and cells [67], where particles representing the RBC membrane experience drag and random forces. Gounley and Peng use the IB machinery to spread membrane viscosity to the fluid, thereby modifying the fluid stress term [73]. Our intent in adding a viscoelastic response is to aid in the numerical stability of the discrete RBCs. We wish to verify that the extended model, with dissipative force, retains the ability to tumble and tank-tread. To that end, we place a single RBC with $n_d = 625$ and $n_s = 2500$ in the same domain as the previous section, now discretized to have $h = 0.4$ µm and with moving top and bottom walls. In the interest of reducing simulation time, we now use the backward-forward Euler timestepping scheme with a time step of $\Delta t = 0.1$ µs. Here, the IB interaction operations

Table 5.1: Convergence of $\boldsymbol{u}$ for a sequence of grids. The refinement ratio r, defined as the refinement in h relative to the coarsest grid, determines the simulation parameters: $rh = 0.8\,\mu\text{m}$ and $r\Delta t = 180\,\text{ns}$. Errors are computed between grids of refinement factor r and $r+1$. Values of $\boldsymbol{u}$ are sampled at $t = 180\,\mu\text{s}$ at cell centers on the coarsest grid.

r	L_2 error	order	L_∞ error	order
1	1.74592×10^{-3}		1.59995×10^{-2}	
2	9.92788×10^{-5}		6.52038×10^{-4}	
3	3.65264×10^{-5}	6.85983	3.34322×10^{-4}	1.06075
4	2.31069×10^{-5}	2.87612	2.30732×10^{-4}	1.60689
5	1.65898×10^{-5}	1.25668	2.28193×10^{-4}	0.83030

Table 5.2: Convergence of $\boldsymbol{X}$ for a sequence of grids. The refinement ratio r, defined as the refinement in h relative to the coarsest grid, determines the simulation parameters: $r\Delta t = 180\,\text{ns}$, $n_d = 125r^2$, and $n_s = 500r^2$. Errors are computed between grids of refinement factor r and $r+1$. Values of $\boldsymbol{X}$ are sampled at $t = 180\,\mu\text{s}$ at $N = 1000$ Bauer spiral points.

r	L_2 error	order	L_∞ error	order
1	6.05447×10^{-3}		2.68249×10^{-3}	
2	1.61678×10^{-3}		6.86140×10^{-4}	
3	7.69150×10^{-4}	2.74664	3.30734×10^{-4}	2.86457
4	4.66203×10^{-4}	1.89474	1.86139×10^{-4}	1.84435
5	2.82578×10^{-4}	1.46574	1.18713×10^{-4}	1.82742

use the 4-point B-spline, $\hat{\delta}_1(r) = B_3(r)$, which was first considered by Lee [77]. It is similar in shape to the Roma kernel but has better smoothness properties. It is shown in Figure 5.1. To recover the tumbling motions, the top wall has a fixed velocity of $\boldsymbol{u}_b = 400\,\mu\text{m/s}$ and the bottom wall $-\boldsymbol{u}_b$. This generates a shear rate of $\dot{\gamma} = 50\,\text{s}^{-1}$ in the absence of cells. For tank-treading experiments, we use $\boldsymbol{u}_b = 8\,\text{mm/s}$ to generate a shear rate of $\dot{\gamma} = 1000\,\text{s}^{-1}$. These values are chosen outside the transitional region between tumbling and tank-treading for the elastic parameters used for the RBC [74]. The velocity field is initially steady for flow without cells. We rotate the cell 1 radian about the x-axis from a horizontally aligned orientation and place it at the center of the domain. The RBC exhibits both of these behaviors; Figure 5.2 shows one period of each.

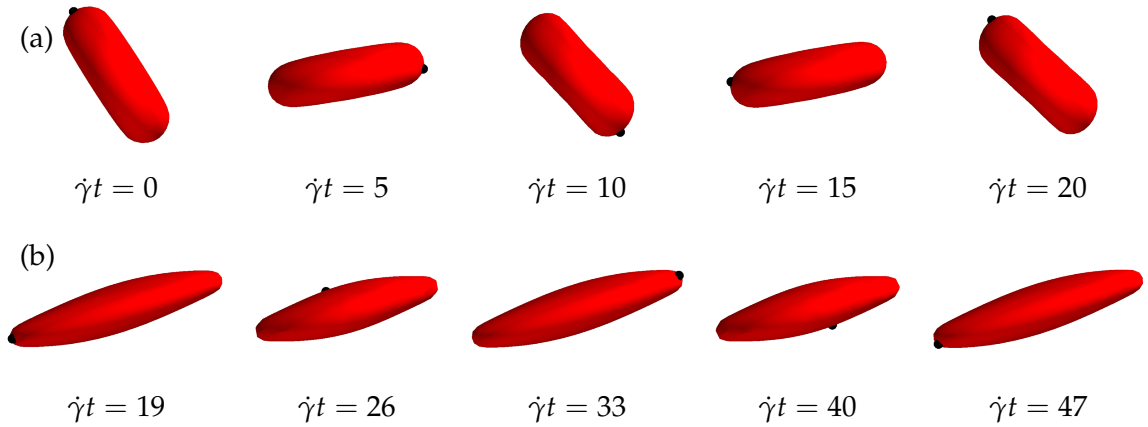


Figure 5.2: Our model RBC exhibits (a) a tumbling behavior under low shear ($\dot{\gamma} = 50\,\mathrm{s}^{-1}$) conditions and (b) tank-treading under high shear ($\dot{\gamma} = 1000\,\mathrm{s}^{-1}$) conditions. The black dot marks a surface point with fixed material coordinates.

5.1.3 Collision tests

With whole blood simulation as the ultimate goal, we must ensure that the method can effectively capture cell-cell interactions. The RBF-IB method has been applied to flow around multiple platelets in an aggregate, but those cells are kept apart by a network of springs [83]. To model the interaction between cells, we devise a series of tests in which we force two RBCs to collide. The aim is to verify that the cells remain distinct. Using too few data sites could allow the cells to come too close to one another. The regularization of δ_h then causes them to be treated as a single unit. Cells that "fuse" in this manner are problematic, generally causing the simulation to end when the cells attempt to separate.

We continue to use the same physical domain as the previous two sections, now with $h = 0.2\,\mu\mathrm{m}$, and place two RBCs therein, each with $n_d = 2500$ and $n_s = 10000$. The ratio $n_s/n_d = 4$ is chosen so that sample sites with spacing h means data sites have spacing $2h$. We believe this to suffice in preventing cells from intersecting, but this is not guaranteed if the points do not maintain appropriate spacing throughout a simulation. We use the 2-stage RK method with time step $\Delta t = 50\,\mathrm{ns}$. To interpolate velocities and spread forces, we use the 4-point cosine $\hat{\delta}_1$ [81], shown in Figure 5.1. The cells are placed with cell centers on the line $x = z$, $y = 8\,\mu\mathrm{m}$. They are initially separated by a gap of $4h = 0.8\,\mu\mathrm{m}$ between their convex hulls, *i.e.*, ignoring the concavities. Inspired by Crowl & Fogelson [66], we add the fictitious force density

$$\boldsymbol{F}_{\text{fict}} = \pm 0.1\,\mathrm{dyn/cm} \cdot (\boldsymbol{e}_1 + \boldsymbol{e}_3)/\sqrt{2},$$

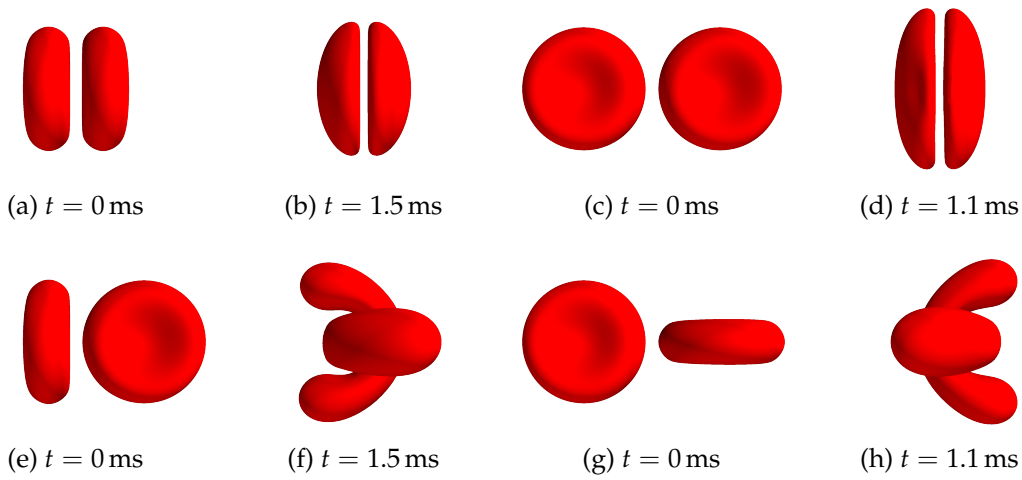
to each cell, where the sign is chosen so the force points into the gap, to draw the cells

together. Success in these tests implies that this configuration of data and sample sites, the spatial resolution, and the time step are acceptable for whole blood simulations.

Initial conditions and configurations after a short time are illustrated in Figure 5.3, where we view them from above the $x = z$ plane. In each case, the cells move slightly closer together and then undergo considerable deformation. The data sites are initially approximately $2h$ apart from each other. No problems seem to arise from this, and in some cases the cells eventually attempt to slide past one another. We also deduce that the IB method with the cosine kernel can resolve interactions at a distance of h to $2h$. We consider cells passing within this threshold to be in contact. Throughout the simulation, the cells remain distinct, and the simulations end due to extreme forces triggering the stopping condition [63]

$$\Delta t > \frac{1}{4}\sqrt{\frac{h\rho}{\|f\|_\infty}}. \tag{5.4}$$

For the remainder of this chapter, we will only consider this arrangement of data and sample sites and this grid resolution.



(a) $t = 0\,\text{ms}$ (b) $t = 1.5\,\text{ms}$ (c) $t = 0\,\text{ms}$ (d) $t = 1.1\,\text{ms}$

(e) $t = 0\,\text{ms}$ (f) $t = 1.5\,\text{ms}$ (g) $t = 0\,\text{ms}$ (h) $t = 1.1\,\text{ms}$

Figure 5.3: Collision tests between two RBCs. A fictitious force is added to the RBCs to draw them together. (a)–(b) The RBCs are initially aligned with concavities facing one another. By 1.5 ms, the cells take on a hemispherical shape. The concavities in the gap are maintained. Shortly thereafter, asymmetries in the setup lead to the cells sliding past one other. (c)–(d) The RBCs are initially aligned with their edges facing one another. By 1.1 ms, the cells take on a hemispherical shape. The remnants of the concavity can be seen on the left cell in (d). Shortly thereafter, these cells also slide past one another. (e)–(f) The RBCs are initially aligned with the edge of one facing a concavity of the other. The cells wrap around each other by 1.5 ms, taking on a bulbous banana shape. (g)–(h) The RBCs are initially aligned with their edges facing one another with one of the cells rotated about the axis $e_1 + e_3$ by $\pi/2$. By 1.1 ms, the cells wrap around each other, again taking on the bulbous banana shape.

5.2 Whole blood

In what follows, we consider a $16\,\mu m \times 12\,\mu m \times 16\,\mu m$ domain with periodic boundaries in the x and z directions and with Dirichlet boundary conditions in the y direction. The fluid velocity is initially zero except at the top boundary, where it moves at 12 mm/s. In the absence of cells, the flow tends toward steady Couette flow with a shear rate of $\dot{\gamma} = 1000\,s^{-1}$. This serves as our model near-wall region of a blood vessel.

For whole blood simulations, we return to the 4-point B-spline, B_3, as the IB kernel. Because we are limited to very small time steps with either timestepping scheme, we use backward-forward Euler with $\Delta t = 50\,ns$ to reduce simulation time. We observe qualitative agreement between these two schemes. We have already settled on an RBC discretization in the previous section. We use the same spiral method to discretize the platelet, but with 900 data and sample sites. Using the same number of sample sites and data sites aligns more closely with traditional IB methods. We also find that the Bauer spiral places points more densely along the edge of the platelet, which is helpful in resolving the large curvatures there. We parametrize the surface of the endothelium over $(\theta, \varphi) \in [0, 2\pi)^2$ with reference shape

$$\hat{X}_{endo}(\theta, \varphi) = \begin{bmatrix} 16\,\mu m \cdot (\theta/2\pi) \\ y(\theta, \varphi) \\ 16\,\mu m \cdot (\varphi/2\pi) \end{bmatrix}, \tag{5.5}$$

where $y(\theta, \varphi)$ depends on the shape under study. The endothelium is discretized using 16000 points along the toroidal spiral described in Appendix B. We consider two shapes for the endothelium. The first, $y = 1\,\mu m$, emulates the flat wall typically seen in near-wall simulations of RBCs or platelets. The other attempts to recreate the elongated endothelial cell shape typical of exposure to high-shear conditions,

$$y(\theta, \varphi) = 0.75\,\mu m + 1\,\mu m \cdot \cos^2(\theta - \varphi)\sin^2(\varphi/2).$$

The bumps have a prominence of 1 µm. The endothelial surface is raised by 0.75 µm to avoid interacting with the domain boundary. The positions of the surface are chosen to maintain a fixed hematocrit of approximately 34% for both endothelial shapes. The bumps in this surface are oriented diagonally across the domain, so we design an initialization process that orients the flow and RBCs in that direction.

As a preliminary validation of the platelet model and to establish baseline platelet motion, we consider two platelets along a flat wall. They are placed parallel to the wall at distances of 0.3 µm and 0.5 µm. The domain does not contain any RBCs. At a distance of

0.3 µm, the platelet is expected to "wobble", in which the platelet tilts slightly upward and downward, periodically [79]. On the other hand, the platelet initially 0.5 µm from the wall should tumble end-over-end. We observe wobbling at a frequency of approximately $10\,\mathrm{s}^{-1}$ and tumbling with a frequency of approximately $30\,\mathrm{s}^{-1}$. We also note that the edge of the tumbling platelet remains pointed towards the wall for only 3–4 ms.

5.2.1 Initialization

We begin by assuming that the platelets have already been marginated by the RBCs. We think of the domain as having three layers with the endothelium at the bottom, RBCs on top, and platelets in between. We begin by settling the endothelium and RBCs before placing platelets.

While the bumps in the endothelium are aligned to flow moving diagonally across the domain, it is much simpler to initially align RBCs for flow moving in the z direction. In addition to the endothelium, we place 2 rows of 4 RBCs, each in their reference configuration, in the domain with its center of mass on the plane $y = 6$ µm and oriented with concavities facing in the z direction. Because the domain is not wide enough to accommodate two reference RBCs alongside one another, the cells are staggered by 2 µm. These locations are then randomly translated and rotated while maintaining a distance of at least $2h$ between cells.

Before placing any platelets in the domain, we allow the flow to develop with only the endothelium and RBCs. The initial fluid velocity is zero except for the top boundary, which has velocity $16\,\mathrm{mm/s} \cdot (\sin\alpha \boldsymbol{e}_1 + \cos\alpha \boldsymbol{e}_3)$. Initially, $\alpha = 0$ so that the flow matches the orientation of the RBCs. Over the course of 7.2 ms, α increases to $\pi/4$ to match the orientation of the endothelium. At this point, the RBCs are not sufficiently mixed. We allow the initialization to continue until at least 17 ms, which is approximately when the first RBC overtakes its neighbor. From here, we choose a series of times, sampled from a Poisson distribution to be approximately 3 ms apart, at which to begin simulations with platelets.

The RBCs for each of the chosen starting configurations are considerably and unpredictably deformed and have left a space of a few microns above the endotheliumin which we place platelets. To find reasonable starting orientations for the platelets, we randomly choose points on the endothelium and one on each platelet surface. Each of the simulations in the upcoming sections contains two platelets, so we choose two points on the endothelium that are at least 3.9 µm, a platelet diameter plus $4h$, apart. The resulting platelet are spaced far enough apart as to not intersect. We compute normal vectors on the surfaces of the endothe-

lium and platelets at these points. The platelets are oriented so that the normal emanating from the platelet opposes the normal at the corresponding point on the endothelium. The platelet is then placed so its chosen surface point is separated from the endothelium point by a random distance between 0.3 µm and 1 µm. If the generated orientation does not pass within 0.4 µm of an RBC, the platelet is accepted. Otherwise, we try again with a different platelet point. This algorithm typically succeeds within 2 attempts.

This initialization process is performed once for each endothelial configuration and we take the first four initial configurations for each. In the following sections, we present behaviors found in these simulations. As a point of comparison, we also consider the initial configurations for the bumpy wall with the RBCs removed from the domain.

5.2.2 Characterization of flow and cell behaviors

In this section, we catalog the differences in the flow between whole blood along a bumpy and flat wall, and between flow along a bumpy wall with and without RBCs. We aim to compare the interactions platelets have with RBCs and the endothelium for these test cases.

Flow profiles are shown in Figure 5.4. The most notable difference among the three flow profiles is the nearly Couette flow when RBCs are absent. The only distinction between this and Couette flow is the smoother transition at the wall, due to the bumps. This is also the distinguishing feature between the profiles corresponding to bumpy and flat walls in the presence of RBCs. The smooth transition from the bumps results in marginally slower flow speeds throughout the domain, compared to the flat wall. The transition in the velocity across the flat wall is comparatively sharp. The inclusion of RBCs causes the bends in the red and blue curves around $y = 3$ µm and $y = 9$ µm. Platelets inhabit the region between $y = 1$ and 3 µm. In simulations featuring RBCs, the platelets experience higher shear rates than those without RBCs due to a reduction in apparent viscosity. This is a consequence of the Fåhræus effect. The bend near $y = 9$ µm is nonphysical and arises from satisfying boundary conditions at the top boundary. However, the increased shear rate in the region between $y = 9$ and 12 µm seems to be useful in deterring RBCs from approaching the upper boundary. An exclusionary region of just 1–2 µm along the top boundary increases the effective hematocrit to 37–41%. Furthermore, the reduced shear rate in the region containing RBCs results in slower tank-treading, with one period now lasting approximately 40 ms.

RBCs are effective at preventing the platelets from moving too far from the endothelium. The furthest observed distance from the endothelium any platelet takes is just under

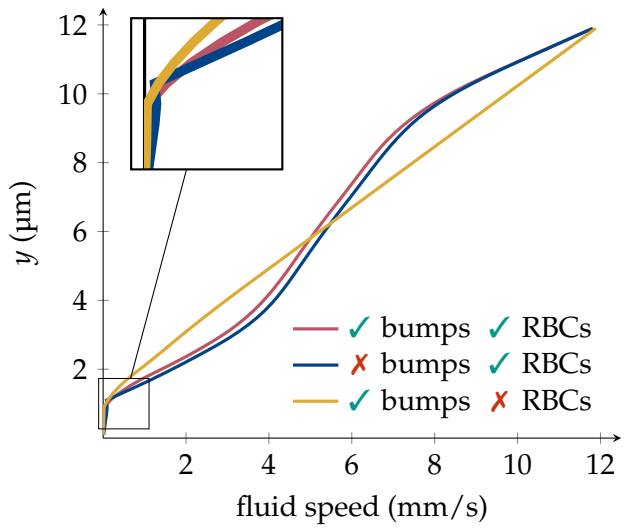


Figure 5.4: Time- and space-averaged fluid velocity profiles for each of the test cases. The inclusion of RBCs (red and blue curves) causes the region inhabited by platelets, where y is approximately between 1 and 4 µm, to experience a higher shear rate than it would without RBCs (yellow curve).

1.5 µm. Likewise, RBCs infrequently enter the cell-free layer, with some notable exceptions, discussed below. We do not observe any platelet wobbling. Instead, platelets transiently follow the curve of the bumpy walls, tilt down into the valleys between bumps, and tumble. Nothing suggests that bumps in the surface of the endothelium alone can sequester platelets, nor do we directly observe stagnation zones.

Bumpy endothelium simulations without RBCs mimic those with a flat wall; platelets move away from the wall to a point where they are free to tumble. Unsurprisingly, we observe platelet tumbling for both flat and bumpy walls with RBCs as well. In Stokes-like flow, a rigid platelet would tumble faster in flow with a higher shear rate. We might therefore expect the platelet to tumble faster with RBCs. However, RBCs can significantly disturb the fluid around a platelet, speeding up its motion, slowing it down, or preventing a tumble altogether.

We observe platelets rolling in the flow direction along their edge. Because the platelet in this arrangement is aligned vertically, part of the edge stays in near-contact with the endothelium while also extending into the region inhabited by RBCs. Contact with RBCs is frequent. These contacts can have a destabilizing effect, but may also prolong the rolling. Figure 5.5(a)–(d) consists of a series of snapshots illustrating the behavior. The platelet in this case is flanked by two RBCs, so it does not have the space to topple over until the RBCs pass a few milliseconds later.

While Figure 5.5 shows this phenomenon on a bumpy wall, it is not limited to that

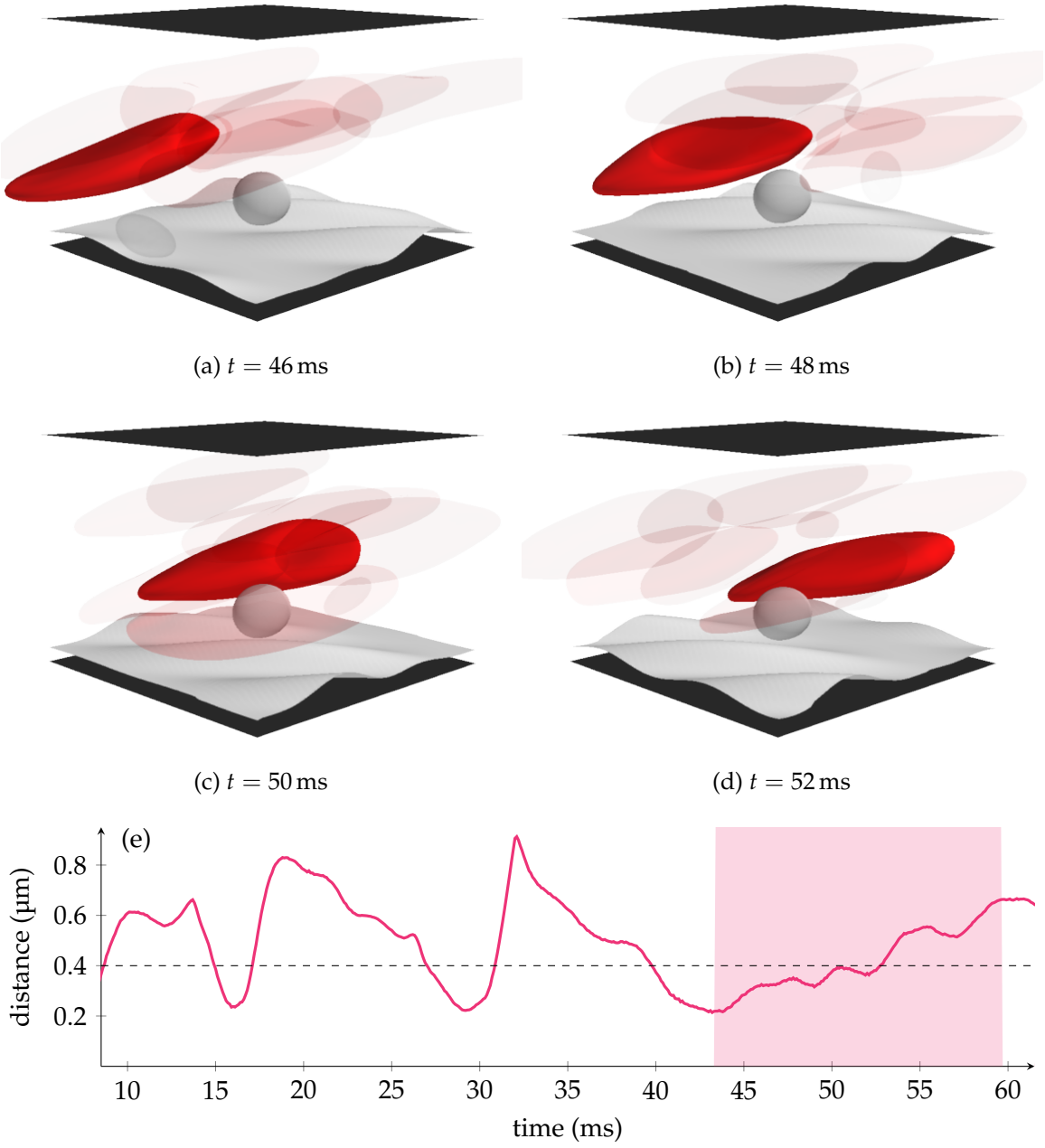


Figure 5.5: An example of a platelet rolling on its edge ("unicycling"). (a)–(d) Snapshots of a platelet unicycling with RBCs, one translucent, flanking either side. The camera tracks the opaque platelet. It's motion is indicated by the endothelium moving from right to left. (e) The distance between the platelet and the endothelium. The shaded region indicates that the orientation of the platelet's short axis is within 45° of the vorticity direction. The black dashed line indicates $2h$ and is the maximum distance that might be considered contact with the endothelium.

endothelial shape. We first observed this motion with a flat wall, and there it lasted over 40 ms. The motion was maintained, in part, by an RBC that rode along the top of the platelet, partially enveloping the platelet. We designate a platelet rolling on its edge as *unicycling*. Figure 5.5(e) illustrates that the platelet spends more time in contact or near-contact with the endothelium while unicycling compared to the tumbles near $t = 16$ ms and $t = 29$ ms. Though RBCs seem to control the duration of the unicycling, they are not strictly necessary for unicycling to occur. In tests with a bumpy wall without RBCs, unicycling is initiated when a platelet rolls sideways, relative to the flow direction, off of a bump. Without RBCs, the platelet maintains the vertical alignment for a majority of the simulation thereafter. However, without frequent interaction with RBCs, the platelet in these simulations move away from the wall. Moreover, while we have not observed it directly, we expect that a lone platelet traveling over a flat wall would also exhibit unicycling, given the right initial orientation.

We notice that in simulations with a bumpy endothelium, platelets will collide with the bumps. This tends to occur while the platelet is tumbling, and the edge of the platelet makes contact with the endothelium. This interaction is characterized by deformations that flatten the edge of the platelet and a ~5% relative change in the aspect ratio of the platelet. However, the collision need not occur along the edge of the platelet, nor, indeed, against a bump in the endothelium. Similar collisions occur in the case of a flat wall, implying that RBCs mediate this behavior. A clear case of this is illustrated in Figure 5.6(a)–(d). We also note that the few milliseconds preceding the unicycling in Figure 5.5 correspond to a collision with the wall, showing that this is yet another trigger for unicycling to occur.

Because the platelet comes into contact with the endothelium, or nearly does so, the platelet slows along the area of contact. Figure 5.6(e) shows the correlation between relative change in aspect ratio and reduction in minimum platelet surface velocity. Though the aspect ratio of the platelet changes somewhat while normally tumbling, changes of this magnitude seem to always correspond to interactions with the endothelium. Collisions with the RBC, for example, result primarily in deformation of the RBC and deflection of the platelet, which is otherwise relatively unperturbed.

5.3 Discussion

We have applied the RBF-IB methodology to the problem of whole blood simulation. RBCs experience contacts in various configurations. Despite using a different number of data and sample sites to discretize them, the RBCs remain distinct. We have shown that a

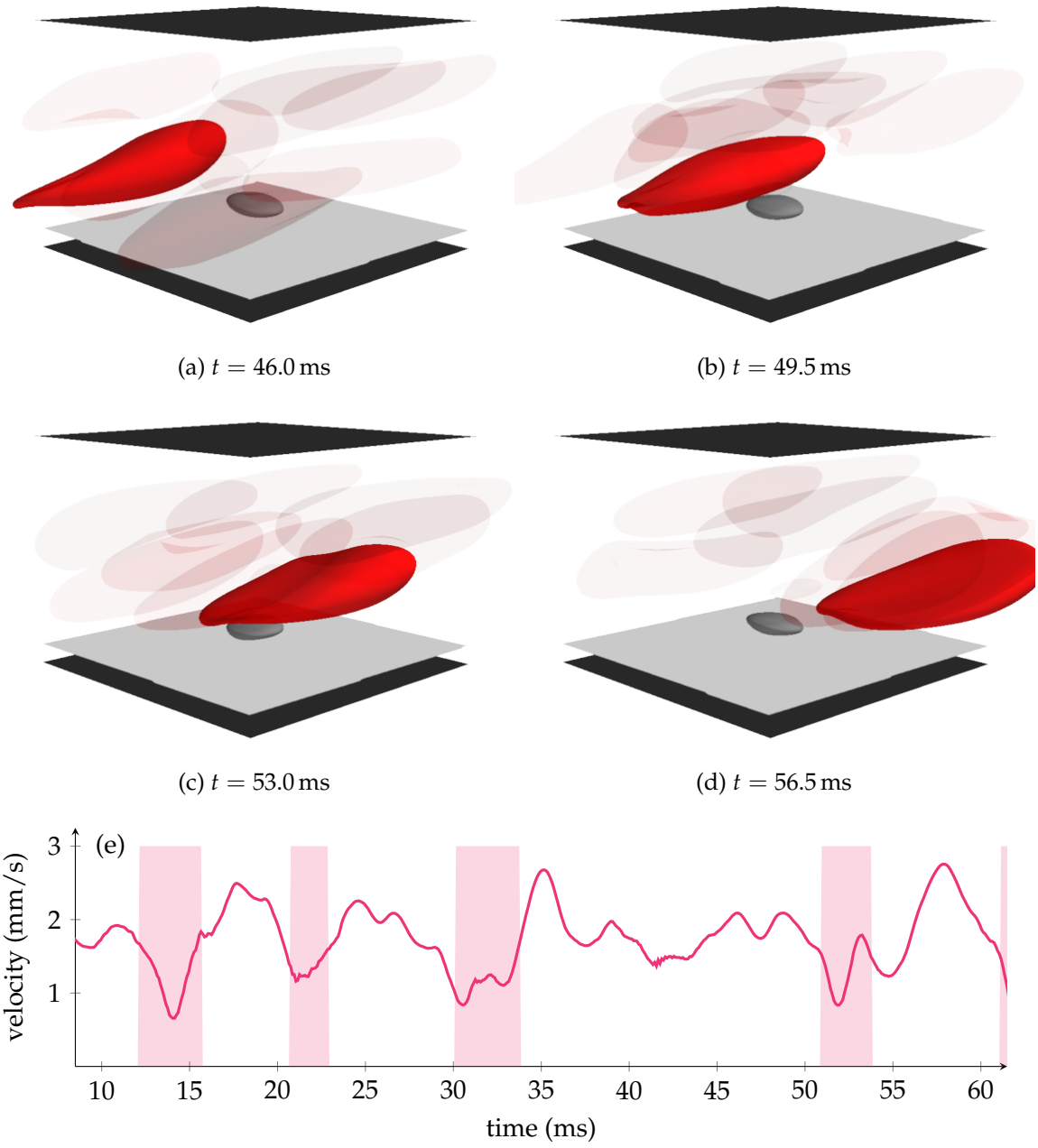


Figure 5.6: An example of an RBC-mediated collision between a platelet and the endothelium. (a)–(d) Snapshots of a collision between a platelet and the endothelium. (a) The platelet attempts to tumble. (b)–(c) An RBC comes into proximity with the platelet, deflects to avoid the platelet, and pushes the platelet into the endothelium, thereby preventing the platelet from tumbling. (d) The platelet is free to tumble again. (e) The minimum velocity on the surface of the platelet. The shaded region indicates that the relative change in aspect ratio exceeds 4%.

continuous energy RBF-based model RBC extended with dissipative forces exhibits tumbling and tank-treading behaviors. We used model RBCs, platelets, and a model endothelium to simulate the flow of whole blood over a perturbed vessel wall. We considered both flat and bumpy endothelial shapes and flow with and without RBCs along a bumpy wall.

The most prominent result of the whole blood simulations is that simulating platelets, but neglecting the influence of RBCs, cannot capture the true nature of platelet motion. Platelets experience augmented shear rates due to RBCs. RBCs confine platelets to the cell-free layer. Simulations without RBCs exhibit regular platelet tumbling or wobbling. In reality, the motion of platelets is more chaotic. Interaction with an RBC would almost certainly disturb a wobbling motion and have been shown to delay tumbling. On a flat wall, certain platelet behaviors would only be observed by considering numerous starting configurations.

The effect of RBCs generally overwhelms the effects of the wall. Figure 5.4 shows that the flat wall yields slightly faster fluid velocities, but the flow profiles for flat and bumpy walls are qualitatively the same, except for the immediate vicinity of the wall. The result is a region of space between the bumps with a velocity gradient. Platelets following the shape of the bumpy wall crest the bump, dip into the region of lower velocity, and tumble. This feature is absent from the flat wall, but tumbling is not an extraordinary behavior. In either case, we observe unicycling, in which the platelet rolls in the flow direction along its edge. Unicycling can be stabilized by RBCs, when RBCs flank the platelet, or by partially encapsulating the platelet as it flows overhead. It can also be destabilized by RBC passing on one side. The endothelial protrusions are also sufficient to orient a platelet in the unicycling, but without RBCs to confine the platelet near the wall, the platelet does not roll along the wall for long. However, we find that unicycling seems to be stable. Unicycling also highlights the need for 3-dimensional simulations—it is a behavior that would not be captured by a 2-dimensional simulation. We also observe platelet-endothelial interactions for both endothelial shapes. These interactions are typically caused by RBCs driving the platelet into the endothelium. The collisions are characterized by significant deformation to the platelet and a speed reduction. From a qualitative standpoint, the endothelial shape alone has minimal impact on the motion of the platelets, meaning that for modeling a healthy blood vessel, a flat wall suffices.

We consider an alternative interpretation of the flat wall: a model exposed subendothelium. Near contact with the subendothelium triggers platelet activation. Unicycling keeps an edge of the platelet near the wall, without hindering mobility. The vertical alignment is

often maintained much longer than wall contacts from tumbling, and we do not observe wobbling in the presence of RBCs. We propose unicycling as an effective means by which platelets survey the vasculature for injury. Of course, the platelet can only distinguish a healthy vessel from an injury by encountering the necessary chemical signals. Until then, the platelets unicycle around bumps along the healthy endothelium as well. This also implies that RBCs indirectly assist in platelet activation for these shear rates.

We also consider an alternative interpretation of the bumpy wall. Because the bumps are approximately the same size as a platelet, we can consider this to be a rough model of a subendothelium with a few deposited platelets. Under this interpretation, we view platelet-wall contact as interactions between an unactivated platelet and the subendothelium, when contact occurs in a valley, or between an unactivated platelet and an between an unactivated and activated platelet adhering to the subendothelium, when contact occurs on a bump. These interactions correlate with a reduction in velocity at the contact zone and the platelet membrane becomes flattened at the point of contact. We suggest that the decreased velocity may be sufficient to allow bonds to form between the platelets or between the platelet and subendothelium. By flattening, the platelet exposes more surface area at the point of contact, so that the activation signals are more likely to reach the platelet. The observed velocity reduction seems insufficient for this, but the resolution of our simulations is also unlikely to allow cells to pass within bonding distance of one another. Overcoming this limitation we leave as a future direction.

References

[63] G. Agresar, J. J. Linderman, G. Tryggvason, and K. G. Powell, *An adaptive, cartesian, front-tracking method for the motion, deformation and adhesion of circulating cells*, Journal of Computational Physics, 143 (1998), pp. 346–380.

[64] R. Bauer, *Distribution of points on a sphere with application to star catalogs*, Journal of Guidance, Control, and Dynamics, 23 (2000), pp. 130–137.

[65] L. C. Erickson and A. L. Fogelson, *Computational model of whole blood exhibiting lateral platelet motion induced by red blood cells*, International Journal for Numerical Methods in Biomedical Engineering, 26 (2010), pp. 471–487.

[66] L. C. Erickson and A. L. Fogelson, *Analysis of mechanisms for platelet near-wall excess under arterial blood flow conditions*, Journal of Fluid Mechanics, 676 (2011), pp. 348–375.

[67] D. A. Fedosov, B. Caswell, and G. E. Karniadakis, *A multiscale red blood cell model with accurate mechanics, rheology, and dynamics*, Biophysical Journal, 98 (2010), pp. 2215–2225.

[68] D. A. Fedosov, J. Fornleitner, and G. Gompper, *Margination of white blood cells in*

microcapillary flow, Physical Review Letters, 108 (2012), p. 028104.

[69] D. A. Fedosov and G. Gompper, *White blood cell margination in microcirculation*, Soft Matter, 10 (2014), pp. 2961–2970.

[70] D. A. Fedosov, H. Noguchi, and G. Gompper, *Multiscale modeling of blood flow: from single cells to blood rheology*, Biomechanics and Modeling in Mechanobiology, 13 (2013), pp. 293–312.

[71] A. L. Fogelson and K. B. Neeves, *Fluid mechanics of blood clot formation*, Annual Review of Fluid Mechanics, 47 (2015), pp. 377–403.

[72] J. B. Freund, *Leukocyte margination in a model microvessel*, Physics of Fluids, 19 (2007), p. 023301.

[73] J. Gounley and Y. Peng, *Computational modeling of membrane viscosity of red blood cells*, Communications in Computational Physics, 17 (2015), pp. 1073–1087.

[74] T. Krüger, M. Gross, D. Raabe, and F. Varnik, *Crossover from tumbling to tank-treading-like motion in dense simulated suspensions of red blood cells*, Soft Matter, 9 (2013), pp. 9008–9015.

[75] A. Kumar and M. D. Graham, *Segregation by membrane rigidity in flowing binary suspensions of elastic capsules*, Physical Review E, 84 (2011), p. 066316.

[76] A. Kumar and M. D. Graham, *Mechanism of margination in confined flows of blood and other multicomponent suspensions*, Physical Review Letters, 109 (2012), p. 108102.

[77] J. H. Lee, *Simulating* in vitro *models of cardiovascular fluid-structure interaction: methods, models, and applications*, PhD thesis, University of North Carolina, Chapel Hill, 2020.

[78] M. Mehrabadi, D. N. Ku, and C. K. Aidun, *Effects of shear rate, confinement, and particle parameters on margination in blood flow*, Physical Review E, 93 (2016), pp. 455–11.

[79] N. A. Mody and M. R. King, *Three-dimensional simulations of a platelet-shaped spheroid near a wall in shear flow*, Physics of Fluids, 17 (2005), p. 113302.

[80] K. Müller, D. A. Fedosov, and G. Gompper, *Margination of micro- and nano-particles in blood flow and its effect on drug delivery*, Scientific Reports, 4 (2014), p. 4871.

[81] C. S. Peskin, *The immersed boundary method*, Acta Numerica, 11 (2002), pp. 479–517.

[82] A. M. Roma, C. S. Peskin, and M. J. Berger, *An adaptive version of the immersed boundary method*, Journal of Computational Physics, 153 (1999), pp. 509–534.

[83] V. Shankar, G. B. Wright, R. M. Kirby, and A. L. Fogelson, *Augmenting the immersed boundary method with radial basis functions (RBFs) for the modeling of platelets in hemodynamic flows*, International Journal for Numerical Methods in Fluids, 79 (2015), pp. 536–557.

[84] T. Skorczewski, L. C. Erickson, and A. L. Fogelson, *Platelet motion near a vessel wall or thrombus surface in two-dimensional whole blood simulations*, Biophysical Journal, 104 (2013), pp. 1764–1772.

[85] K. Vahidkhah and P. Bagchi, *Microparticle shape effects on margination, near-wall dynamics and adhesion in a three-dimensional simulation of red blood cell suspension*, Soft Matter, 11 (2015), pp. 2097–2109.

[86] K. Vahidkhah, S. L. Diamond, and P. Bagchi, *Platelet dynamics in three-dimensional simulation of whole blood*, Biophysical Journal, 106 (2014), pp. 2529–2540.

[87] W. Wang, T. G. Diacovo, J. Chen, J. B. Freund, and M. R. King, *Simulation of platelet, thrombus and erythrocyte hydrodynamic interactions in a 3D arteriole with in vivo comparison*, PLoS ONE, 8 (2013), p. e76949.

[88] Z. Wu, Z. Xu, O. V. Kim, and M. S. Alber, *Three-dimensional multi-scale model of deformable platelets adhesion to vessel wall in blood flow*, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 372 (2014), p. 20130380.

[89] H. Zhao and E. S. G. Shaqfeh, *Shear-induced platelet margination in a microchannel*, Physical Review E, 83 (2011), p. 061924.

[90] H. Zhao, E. S. G. Shaqfeh, and V. Narsimhan, *Shear-induced particle migration and margination in a cellular suspension*, Physics of Fluids, 24 (2012), pp. 011902–011921.

CHAPTER 6

SUMMARY AND FUTURE WORK

6.1 Summary

In this dissertation, we describe a method for using radial basis function (RBF) surface interpolation to compute surface forces for general constitutive laws. We present advances in immersed boundary (IB) method parallelization that enable the implementation of RBF-based geometry for the IB (RBF-IB) method entirely on the GPU. Ultimately, we apply these techniques to the problem of simulating whole blood. We give background for each of the cellular components included in our simulations and the biological motivation in Chapter 1.

We continue to Chapter 2 for a view of the standard IB method, starting with the fluid equations. We give details of the discretization of the incompressible Navier-Stokes equations and describe the various elastic and viscoelastic energy models used in representing blood cells and the endothelium. We detail the discretization of the blood cells and the RBF-based methods for reconstructing the cell surfaces from a point cloud. We use the discretization to construct discrete linear operators which allow us to recover geometric data for the cell: tangents, their derivatives, and quadrature weights for the sphere using RBFs—a preliminary to quadrature weights for the cell. The method allows us to use a different number of points to recreate the cell than are used to evaluate forces. Finally, we present the IB method, which we use to couple the cellular and fluid components of blood.

In Chapter 3, we introduce a discretization of cellular surface forces for smoothly reconstructed surfaces. It uses the analytic force density in terms of geometric quantities on the surface. Equipped with the ability to approximate the surface geometry terms from Chapter 2, we describe a method to evaluate the force density. Moreover, we use surface geometry to construct surface quadrature weights from spherical quadrature weights. With appropriate quadrature weights, we recover forces from the computed force density. We use spherical harmonics to compare this method to forces derived from piecewise linear surface interpolation. The continuous representation gives higher quality forces than its piecewise counterpart. We apply the discretization to the problems of cell blebbing and a single RBC passing through a narrow tube in an unsteady Stokes flow context. With sufficiently many

interpolation points, the continuous representation is on par with traditional IB methods in preventing leakage through a single cell membrane.

Library routines provide parallelization for the fluid solver and surface geometry operators in Chapter 2. In Chapter 4, we develop algorithms to parallelize the IB interaction operations. We present a straightforward parallelization of the velocity interpolation operator. To avoid the potential write contentions that have typically prevented effective parallelization of the spreading operation, we decompose the problem into applications of the key-value sort and segmented reduce parallel primitives. This makes the algorithms suitable for either multicore CPUs or GPUs. Unlike previous attempts at parallelization, these algorithms are simultaneously independent of the fluid grid and capable of concurrently processing arbitrary Lagrangian points. We develop two variant algorithms that trade-off dependence on the fluid grid for increased throughput for small grids. The algorithms exhibit near-perfect scaling with increasing problem sizes and increasing computational resources and expect even better performance from more capable hardware. We now have a complete RBF-IB implementation entirely on the GPU.

Chapter 5 represents the culmination of all of our work: three-dimensional whole blood simulations. We begin by validating that the red blood cell (RBC) model and RBF-IB work as intended by establishing convergence, recovering typical RBC behaviors, and verifying that cells remain distinct when in close contact. For the tests presented, this lends further credence to the idea that using a different number of points to move an IB structure than is used to evaluate forces can avoid leakiness, now in the case of multiple interacting cells. We simulate whole blood flow over a bumpy and flat wall and RBC-depleted blood over a wavy wall. The simulations involving RBCs are qualitatively the same, but comparison with the tests without RBCs reveals that platelet motion is only accurately captured by considering the effects of RBCs. We observe two stand-out behaviors: unicycling, in which a platelet rolls along the wall on its edge, and RBC-mediated collisions between platelets and the wall. Unicycling accompanies periods in which the platelet remains near the wall. We propose this as a mechanism by which the platelet surveys the wall for damage. Collisions, on the other hand, bring with them a reduction in velocity and platelet deformation along the region of contact. The reduction in velocity may suffice to allow chemical signals from a budding aggregate to reach the unactivated platelet. If so, the deformation aids in accelerating the transmission of chemical signals by exposing more platelet surface area to the wall.

6.2 Future work

While this dissertation answers some questions, it raises others. This final section discusses potential numerical, computational, and biological extensions of the presented work. We consider replacing the global surface reconstructions with local interpolants to more closely follow typical implementations of the IB method, an adaptation of the parallel interpolation and spreading IB operations to distributed memory architectures, and an experiment in modeling endothelial cells.

6.2.1 RBF methods for traditional IB

Implementations of the IB method typically use the same set of Lagrangian points to perform the spreading and interpolation operations. Operators with this property are adjoints, which implies that the operations discretely conserve energy [100]. As stated in Section 2.3, we use a set of data sites for RBC surface interpolation, and a different set of sample sites from which to spread forces, to reduce the memory footprint of the discrete differential operators. However, the bending force density (3.81) contains $\nabla_s^2 H$, which we compute by first approximating H at data sites and then applying the discrete surface Laplacian evaluated at sample sites. This requires two sets of discrete operators. Though we still achieve our goal of reducing memory usage for cell representations, using a single Lagrangian grid drastically simplifies the cell representation.

There are additional concerns with using a smaller set of data sites. In high shear rate regimes, RBCs exhibit the tank-treading behavior, even in relative proximity to other RBCs. In 2 dimensions, this underpins the behavior described as "tank-trading" [97]. Suppose that the data sites are, say, $3h$ apart from their nearest neighbors. As neighboring RBCs tank-tread, their data sites continually move. It appears likely that a data site on one of the cells will eventually find a gap in the data sites of the other. Even a small gradient in the fluid velocity between the two cells can lead to one cell encroaching on the other until the cells become entangled. This manifests as a sharp spike in the surface of one or both cells, as they attempt to pull themselves apart, unsuccessfully. Nearly all simulations with neighboring, tank-treading RBCs fail due to this issue. The simulations in Chapter 5 reduce the data site spacing to $2h$ and the domain to constrain the motion of the RBCs in an attempt to combat this issue. We have also mimicked IBFE [100] by tracking sample sites and solving a least-squares problem to reconstruct the surface at data sites. This leads to unstable linear operators in the force calculations. RBCs constructed in this manner exhibit unsustainable wrinkling. Even operators used for stabilization such as hyperviscoscosity [101] have

spurious, unstable eigenvalues.

Other RBF-based methods exist which use local interpolants but still generate a high-fidelity reconstruction of the surface. The resulting discrete linear operators are sparse. The idea is to use the same set of points to construct the local interpolants and evaluate forces. The sparsity of the linear operators drastically reduces their memory footprint compared to their global counterparts. The first of these methods is RBF-based finite differences (RBF-FD), which constructs a piecewise, high-order interpolant. Whereas we have avoided the singularities in using a global coordinate system on the sphere by discretizing away from the poles, local RBF-FD interpolants have no such limitation. The other method is the RBF partition of unity method (RBF-PUM), which stitches together local RBF-FD interpolants into a global interpolant with minimal additional fill-in of the sparse linear operators. Cursory attempts at replacing the global interpolants of Section 2.3 by either RBF-FD or RBF-PUM result in unexplained membrane deformations not exhibited by the global interpolant. With these deformations come nonphysical forces, which terminate the simulation.

Solving this problem requires understanding the differences between piecewise linear, global RBF, and piecewise RBF interpolants. Curiously, the high-order global RBF and low-order piecewise linear interpolants seem to work well for certain types of problems. Piecewise linear interpolants are extremely tolerant of deformation, in part because they do not accurately recover force densities. RBF-FD and RBF-PUM may therefore fill the niche of accurately computing force while effectively preventing overlap between cells.

6.2.2 A distributed memory parallel IB method

The conclusions of this thesis are limited by our ability to simulate whole blood with sufficient speed and resolution. Part of this is due to the extreme time steps required to maintain stability. This, in turn, is likely due to the timestepping method. Backward-forward Euler, for example, has severe time step restrictions under advection-dominated conditions [91]. Methods with more permissive time step allowances tend to require spreading forces multiple times in a timestep. Fortunately, this dissertation provides a fast algorithm for spreading. However, we observe a bottleneck at the Poisson solve required to satisfy incompressibility, taking up approximately 60% of the wall clock time. For compute-bound components, one possibility for solving this issue is to distribute the work among several devices. Libraries such as `hypre` [99], `PETSc` [92], and `AmgX` [104] provide distributed linear solvers.

This dissertation does not consider how Algorithms 4.3 and 4.4–4.6 will perform on

distributed systems, though we anticipate they will perform well, as they do not rely on partitioning the Eulerian or Lagrangian points. Because information about the Eulerian grid is encoded by the functions κ and #, an optimized distributed-memory algorithm will use a different κ or # function on each node, to account for Eulerian grid partitioning. Partitioning the Eulerian grid also reduces the number of keys needed to represent the portion of a grid assigned to a node. So, while the number of grid cells increases, the number of grid cells assigned to a single node stays below 2^{32}, and the conditions for linear runtime, as described in Section 4.1.3, are satisfied. As a side effect, we remain in the regime where the buffered spreading variants, outperform their bufferless alternative.

Distributing the Lagrangian structures is the final hurdle to overcome. A Lagrangian structure that spans Eulerian grid partitions must either (a) be copied in its entirety to both nodes, or (b) divided among the nodes and communicate partial geometric data across nodes. Both methods require some amount of communication to perform velocity interpolation. Implementing (a) with parallel spreading and interpolation requires some simple preprocessing to avoid duplicating work. However, (b) is incompatible with the global cell interpolants presented here, as no node would have the information required to compute complete geometric data. This is ameliorated, in part, by RBF-FD methods, but (a) likely requires less communication, and is currently the method of choice [96, 100]. As simulations become more computationally intensive, distributed algorithms become increasingly necessary.

6.2.3 Models of endothelial cells

The shape of an endothelial cell is determined primarily by its adhesion to the subendothelium, its nucleus, and exposure to shear stress [95, 102, 103]. They take on a cobblestone arrangement under low shear conditions and elongate in high shear. Their nuclei are displaced by flow [105]. Caille *et al.* subjected individual endothelial cells to compression tests to estimate their mechanical properties [93]. They find that the nucleus is much more resistant to deformation than the cytosol when treated as Mooney-Rivlin solids, but that the cytosol behaves as a protective barrier to the nucleus.

We have taken some liberties in modeling the endothelium, following [94] in giving it a sinusoidal shape. As a preliminary to more realistic models of the endothelium, we devise a simple exploration of endothelial dynamics with the hope of recovering some of the observed behaviors. Treat the endothelium as a relatively rigid nucleus embedded within a deformable cell membrane. Along the bottom of the cell, we tether the cell to a flat

subendothelium with breakable tethers. A diagram of the setup is shown in Figure 6.1. The neo-Hookean energy density is a reasonable choice for the cell and nuclear membranes due to its generality and because we know how it behaves for shear moduli in the range between that of an RBC [98] and a platelet. There are ostensibly only 5 parameters to consider: the shear and bending moduli for the cell and nuclear membranes, and the volume of the cell. With an improved model of the endothelial cell, we move ever closer to realistic blood simulation.

References

[91] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, *Implicit-explicit runge-kutta methods for time-dependent partial differential equations*, Applied Numerical Mathematics, 25 (1997), pp. 151–167.

[92] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc Web page*, 2021, https://www.mcs.anl.gov/petsc.

[93] N. Caille, O. Thoumine, Y. Tardy, and J.-J. Meister, *Contribution of the nucleus to the mechanical properties of endothelial cells*, Journal of Biomechanics, 35 (2002), pp. 177–187.

[94] C. Chung, J. K. Chang, B. G. Min, and D. C. Han, *Streamlined shape of endothelial cells*, KSME International Journal, (2010), pp. 861–866.

[95] C. F. Dewey Jr, S. R. Bussolari, M. A. Gimbrone Jr, and P. F. Davies, *The dynamic response of vascular endothelial cells to fluid shear stress*, Journal of Biomechanical Engineering, 103 (1981), pp. 177–185.

[96] L. C. Erickson, *Blood flow dynamics: a lattice Boltzmann immersed boundary approach*, PhD thesis, University of Utah, Salt Lake City, Dec. 2010.

[97] L. C. Erickson and A. L. Fogelson, *Analysis of mechanisms for platelet near-wall excess under arterial blood flow conditions*, Journal of Fluid Mechanics, 676 (2011), pp. 348–375.

[98] T. G. Fai, B. E. Griffith, Y. Mori, and C. S. Peskin, *Immersed boundary method for variable viscosity and variable density problems using fast constant-coefficient linear solvers I: numerical method and results*, SIAM Journal on Scientific Computing, 35 (2013), pp. B1132–B1161.

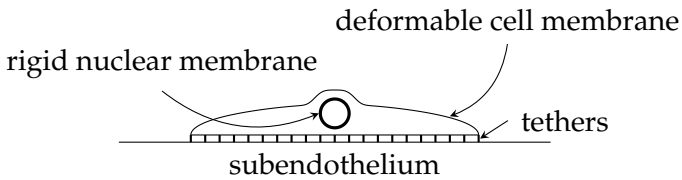


Figure 6.1: Endothelium modeled as a rigid nucleus within a deformable cell membrane, which is tethered along its bottom to the subendothelium.

[99] R. Falgout, A. Barker, T. Kolev, R. Li, S. Osborn, D. Osei-Kuffuor, V. Paludetto Magri, J. Schroder, B. Sjogreen, P. Vassilevski, and U. Meier Yang, *HYPRE*, 2021, https://computing.llnl.gov/projects/hypre.

[100] B. E. Griffith and X. Luo, *Hybrid finite difference/finite element immersed boundary method*, International Journal for Numerical Methods in Biomedical Engineering, 33 (2017), p. e2888.

[101] E. Larsson, E. Lehto, A. Heryudono, and B. Fornberg, *Stable computation of differentiation matrices and scattered node stencils based on gaussian radial basis functions*, SIAM Journal on Scientific Computing, 35 (2013), pp. A2096–A2119.

[102] M. J. Levesque, D. Liepsch, S. Moravec, and R. M. Nerem, *Correlation of endothelial cell shape and wall shear stress in a stenosed dog aorta.*, Arteriosclerosis, Thrombosis, and Vascular Biology, 6 (1986), pp. 220–229.

[103] M. J. Levesque and R. M. Nerem, *The elongation and orientation of cultured endothelial cells in response to shear stress*, Journal of Biomechanical Engineering, 107 (1985), pp. 341–347.

[104] NVIDIA, P. Vingelmann, and F. H. Fitzek, *CUDA*, 2021, https://developer.nvidia.com/cuda-toolkit.

[105] E. Tkachenko, E. Gutierrez, S. K. Saikin, P. Fogelstrand, C. Kim, A. Groisman, and M. H. Ginsberg, *The nucleus of endothelial cell as a sensor of blood flow direction*, Biology Open, 2 (2013), pp. 1007–1012.

APPENDIX A

BOUNDARY ERROR CORRECTION FOR STAGGERED GRIDS

The marker-and-cell (MAC) grid [106] is a popular method for fluid simulations. Components of vector-valued quantities are discretized at the center of the corresponding cell faces and scalar-valued quantities at the cell center. This staggering is a distinguishing feature of the MAC grid. Staggering avoids the checkerboard instability that arises from using collocated grids [107]. However, in domains with nonperiodic boundaries, this means that some vector components will encounter situations where satisfying boundary conditions with linear ghost value extrapolation leads to numerical error. This appendix explores these errors and provides a resolution that maintains compatibility with the conjugate gradients method.

A.1 A simple case

To illustrate the need for boundary correction, we consider a simplified problem. Because every linear solve in the fluid solver depends on some form of the discrete Laplacian, we focus our attention on that operator. Consider the 1-dimensional diffusion problem for $u = u(x, t)$,

$$u_t = \mu u_{xx} + f \qquad \text{for } x \in (0, 1), \tag{A.1}$$

$$\gamma_0 = \alpha_0 u + \beta_0 u_x \quad \text{at } x = 0, \tag{A.2}$$

$$\gamma_1 = \alpha_1 u - \beta_1 u_x \quad \text{at } x = 1, \tag{A.3}$$

where $\alpha_m^2 + \beta_m^2 \neq 0$ for $m = 0, 1$. Here, subscripts t and x indicate partial differentiation with respect to time and space, respectively. We discretize $[0, 1]$ into N cells of length $h = 1/N$ and let u_i^n approximate $u(x_i, n\Delta t)$ at $x_i = h(g + i - 1)$, where i ranges over $\{1, \ldots, N\}$, time $t = n\Delta t$, and $g \in (0, 1]$ is the one-dimensional grid staggering. Consider the boundary at $x = 0$. Let $x_0 = h(g - 1)$ be a ghost point to the left of the boundary, and let u_0^n be the extrapolated value of u at the ghost point. Near the boundary, we use the linear

approximations

$$u(0, n\Delta t) \approx a u_1^n + b u_g^n \quad \text{and} \quad u_x(0, nk) \approx a' u_1^n + b' u_g^n. \tag{A.4}$$

We expect a' and b' to be of order $\mathcal{O}(h^{-1})$, and require that these approximations be at least first order:

$$a + b = 1, \quad a' + b' = 0, \quad \text{and} \quad a'hg + b'h(g-1) = 1. \tag{A.5}$$

For simplicity, we drop the subscripts from α_0, β_0, and γ_0, and the superscripts indicating the timestep. Substituting into the boundary condition yields

$$\gamma = \alpha(au_1 + bu_0) + \beta(a'u_1 + b'u_0) + \mathcal{O}(h). \tag{A.6}$$

Given a value u_1 and boundary data γ, we can extrapolate

$$u_0 \approx (\alpha b + \beta b')^{-1}(\gamma - (\alpha a + \beta a')u_1), \tag{A.7}$$

when $\alpha b + \beta b' \neq 0$. In the extraordinary case that this does not hold, the boundary condition is of Robin type, with neither α nor β zero, and value at the ghost point is arbitrary. We then have four linearly independent conditions for the weights a, b, a', and b': the three conditions in (A.5) and $\alpha b + \beta b' = 0$. Solving for the weights reduces (A.6) to $\alpha u_1 = \gamma$. We do not consider this case further.

Assuming $\alpha b + \beta b' \neq 0$, the standard 3-point discrete Laplacian at x_1 gives the approximation

$$u_0 - 2u_1 + u_2 = (\alpha b + \beta b')^{-1}\gamma - \left(2 + (\alpha b + \beta b')^{-1}(\alpha a + \beta a')\right) u_1 + u_2. \tag{A.8}$$

Replacing approximations with exact values and Taylor expanding about x_1 yields

$$\begin{aligned} u_0 - 2u_1 + u_2 &= (\alpha b + \beta b')^{-1} \left(\alpha \left(u - hgu_{xx} + 1/2(hg)^2 u_{xx}\right) + \beta(u_x - hgu_{xx})\right) \\ &\quad - \left(2 + (\alpha b + \beta b')^{-1}(\alpha a + \beta a')\right) u + \left(u + hu_x + 1/2h^2 u_{xx}\right) + \mathcal{O}(h^3) \\ &= (\alpha b + \beta b')^{-1}(\alpha(1 - a - b) - \beta(a' + b'))u \\ &\quad + (\alpha b + \beta b')^{-1}(\beta - \alpha hg + (\alpha b + \beta b')h)u_x \\ &\quad + (\alpha b + \beta b')^{-1} \left(1/2\alpha(gh)^2 - \beta hg + 1/2(\alpha b + \beta b')h^2\right) u_{xx} + \mathcal{O}(h^3). \end{aligned} \tag{A.9}$$

The coefficient of u vanishes according to (A.5). We further require that the coefficient of u' be zero. That is,

$$\alpha b + \beta b' = -h^{-1}(\beta - \alpha hg).$$

Choosing $a = 1 - g$, $b = g$, $a' = h^{-1}$, and $b' = -h^{-1}$ satisfies these conditions. As a result,

$$\alpha a + \beta a' = h^{-1}(\beta + \alpha h(1 - g)).$$

Finally, the first possibly nonzero coefficient is that of u_{xx}:

$$\begin{aligned} h^{-2}\left[u_0 - 2u_1 + u_2\right] &= \left[1/2 + g(\beta - \alpha hg)^{-1}(\beta - 1/2hg)\right] u_{xx} + \mathcal{O}(h^2) \\ &= \left[1 - 1/2(\beta - \alpha hg)^{-1}(\beta(1 - 2g) - \alpha hg(1 - g))\right] u_{xx} + \mathcal{O}(h^2) \\ &:= (1 - \epsilon)u_{xx} + \mathcal{O}(h^2). \end{aligned} \tag{A.10}$$

Near the boundary, when $\epsilon \neq 0$, *i.e.*, $\beta(1 - 2g) - \alpha hg(1 - g) \neq 0$, these approximations yield a 0^{th}-order approximation to the Laplacian. Cases where $\epsilon \neq 0$ arise naturally from using staggered grids in a domain with at least one nonperiodic dimension. In fact, for fixed α and β, only $g = g^*(2\beta/\alpha h)$ results in $\epsilon = 0$, where

$$g^*(r) = \begin{cases} 1/2\left(1 + r + \sqrt{1 + r^2}\right), & r \leq 0 \\ 1/2\left(1 + r - \sqrt{1 + r^2}\right), & r > 0. \end{cases}$$

For Neumann boundaries, $\epsilon = 0$ when $g = 0.5$; for Dirichlet boundaries, when $g = 1$. The case for the opposing boundary is very similar: simply substitute the correct boundary condition coefficients and data, $-h$ for h, and when $g \neq 1$, $1 - g$ for g. We will use ϵ_0 and ϵ_1, when necessary, to distinguish the error factor when approximating the Laplacian at the $x = 0$ and $x = 1$ boundaries, respectively.

Suppose that in approximating the solution to (A.1)–(A.3), we employ the Crank-Nicolson IMEX timestepping scheme. The discrete equations are

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\mu}{2}\left(\frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{h^2} + \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2}\right) + f_i^{n+1/2}, \tag{A.11}$$

where superscripts denote the time step and subscripts the space step. With $\lambda = \mu\Delta t$, we rewrite this in matrix form as

$$(I - 1/2\lambda\nabla_h^2)\boldsymbol{u}^{n+1} = (I + 1/2\lambda\nabla_h^2)\boldsymbol{u}^n + \lambda B_h\boldsymbol{\gamma}^{n+1/2} + \Delta t\boldsymbol{f}^{n+1/2}, \tag{A.12}$$

where B_h modifies equations near the boundary with boundary data, according to (A.8). As we have shown, this introduces an error near the boundary. This error will propagate into the center of the domain at a rate dependent upon μ. Figure A.1 illustrates this phenomenon. The convergence test in Table A.1 shows that these errors vanish at second order for Dirichlet boundary conditions. While Dirichlet boundary errors vanish at second order, the method converges to the wrong steady state. It is clear from Figure A.1(b) that despite the first-order convergence shown in Table A.2, Neumann boundary errors will only grow as the simulation progresses.

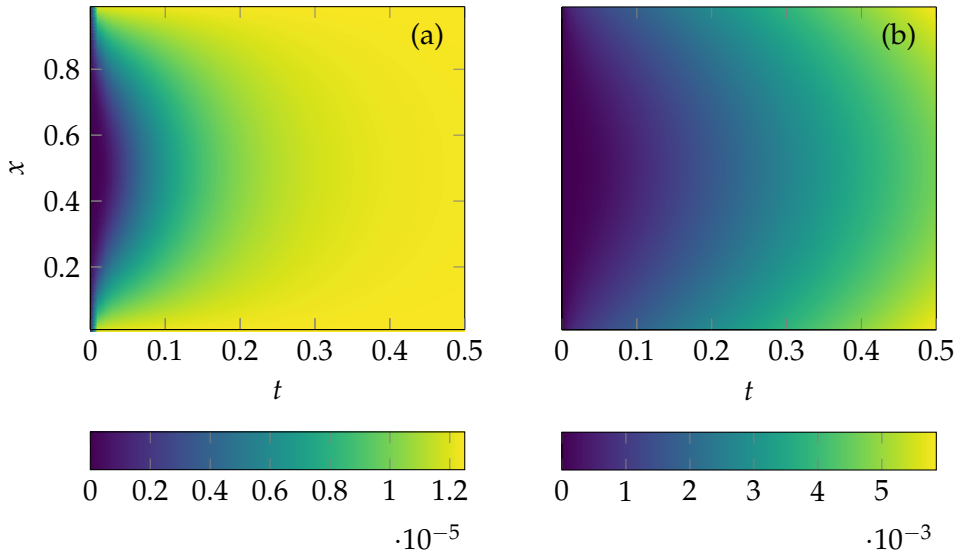


Figure A.1: Propagation of errors near the boundary in approximating the solution of $u_t = u_{xx} + 1$ on $[0, 1]$ without correction at the boundary. Initially, u is analytically steady: $u(x, 0) = x(1 - x)/2$. We expect no change in u over time. (a) u satisfies homogeneous Dirichlet boundary conditions. The domain is discretized using $h = 0.01$, with points $x_i = h(i - 0.5)$ for $i = 1, \ldots, 100$. (b) u satisfies the Neumann boundary conditions $u_x(0, t) = -u_x(1, t) = 1/2$. The domain is discretized using $h = 0.01$, with points $x_i = hi$ for $i = 1, \ldots, 99$.

Table A.1: Convergence test for Crank-Nicolson timestepping without boundary correction for the test problem $u_t = u_{xx} + 1$ with initial steady conditions $u(x, 0) = x(1 - x)/2$ and homogeneous Dirichlet boundary conditions.

N	Δt (ms)	$\|u - u_0\|_2$	order	$\|u - u_0\|_\infty$	order
25	4	142.438496		194.026473	
50	2	35.639706	1.99878	49.254058	1.977949
100	1	8.911804	1.99969	12.406781	1.989114
200	0.5	2.228068	1.99992	3.113348	1.994590

A.2 One-dimensional correction

To improve the approximations near the boundary, we replace the corresponding rows of (A.12) with those obtained by discretizing the scaled equation

$$(1 - \epsilon)u_t = \mu(1 - \epsilon)u_{xx} + (1 - \epsilon)f. \tag{A.13}$$

The solution should be identical to that of the original equation as long as $\epsilon \neq 1$, but the Laplacian constructed above need not be modified to approximate $(1 - \epsilon)u_{xx}$. We simply

Table A.2: Convergence test for Crank-Nicolson timestepping without boundary correction for the test problem $u_t = u_{xx} + 1$ with initial steady conditions $u(x, 0) = x(1-x)/2$ and Neumann boundary conditions $u_x(0, t) = -u_x(1, t) = 1/2$.

N	Δt (ms)	$\|u - u_0\|_2$	order	$\|u - u_0\|_\infty$	order
25	4	4308.714364		6950.600999	
50	2	2141.879730	1.00838	3559.181576	0.965592
100	1	1067.893238	1.00411	1801.330925	0.982482
200	0.5	533.192720	1.00204	906.192341	0.991174

multiply the remaining terms by $1 - \epsilon$. Define the modified identity matrix

$$\tilde{I} = \begin{bmatrix} 1-\epsilon_0 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1-\epsilon_1 \end{bmatrix}. \tag{A.14}$$

Rescaling equations for values near the boundary, equation (A.12) becomes

$$(\tilde{I} - 1/2\lambda\nabla_h^2)\boldsymbol{u}^{n+1} = (\tilde{I} + 1/2\lambda\nabla_h^2)\boldsymbol{u}^n + \lambda B\boldsymbol{\gamma}^{n+1/2} + \Delta t\tilde{I}\boldsymbol{f}^{n+1/2}. \tag{A.15}$$

This improves the error near the boundary to second-order at the cost of one more diagonal matrix-vector multiplication.

Alternatively, one could approximate the Laplacian near the boundary using a quadratic interpolant. It would always give a second-order approximation but would break the symmetry of the discrete Laplacian. Linear interpolation maintains symmetry, and the coefficients obtained near the boundary are exactly those of the quadratic interpolant, scaled by $1 - \epsilon$. The correction recovers the solutions to the problems illustrated in Figure A.1 to machine precision. By modifying only the identity matrix, the discrete Helmholtz operator, $\tilde{I} - 1/2\lambda\nabla_h^2$, maintains its symmetry, and $\epsilon < 1$ is sufficient to maintain positive-definiteness. Many types of boundary conditions satisfy $\epsilon < 1$, most notably all boundary conditions of Dirichlet or Neumann type. We can therefore continue to use conjugate gradients for the linear solves.

A.3 Higher-dimensional correction

We now consider a higher-dimensional Laplacian. Construction, and therefore correction, proceeds recursively, by analog to the continuous Laplacian. For example, the

three-dimensional Laplacian is the sum of the two-dimensional Laplacian in x and y and the second-derivative operator with respect to z. The discrete analog of adding operators is the tensor sum, e.g.,

$$L_y \oplus L_x = I_y \odot L_x + L_y \odot I_x,$$

where L_x and L_y are square, one-dimensional discrete second-derivative operators with respect to x and y, respectively; I_x and I_y are identity operators the same size as L_x and L_y, respectively; and $\odot$ is the Kronecker tensor product. The Kronecker tensor product takes two square matrices, $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{m\times m}$, and produces the $mn \times mn$ block matrix

$$A \odot B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}B & a_{n2}B & \dots & a_{nn}B \end{bmatrix}.$$

If A and B are also symmetric, so is $A \odot B$. The two-dimensional discrete identity operator is also constructed via tensor product: $I_y \odot I_x$. The k-dimensional discrete Laplacian and identity operators are computed recursively via

$$\begin{aligned} L^{[k]} &= I^{[1]} \odot L^{[k-1]} + L^{[1]} \odot I^{[k-1]} = L^{[1]} \oplus L^{[k-1]}, \\ I^{[k]} &= I^{[1]} \odot I^{[k-1]}, \end{aligned} \tag{A.16}$$

where the superscript indicates the dimensionality of the operators.

Consider a two-dimensional diffusion problem on the domain $\Omega = [0, 1]^2$,

$$u_t = \mu \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) u + f \quad \text{in } \Omega, \tag{A.17}$$

$$\gamma = \mathcal{B}u \qquad\qquad \text{on } \partial\Omega, \tag{A.18}$$

where $\mathcal{B}$ is a boundary operator. We imagine the case where either one of the second derivatives composing the Laplacian requires correction at the boundary. Without loss of generality, we will assume they both do. We scale Equation (A.17) as we did in Equation (A.13):

$$(1 - \epsilon_x)u_t = \mu \left(\frac{\partial^2}{\partial x^2} + (1 - \epsilon_x)\frac{\partial^2}{\partial y^2} \right) u + (1 - \epsilon_x) f \tag{A.19}$$

near an x boundary,

$$(1 - \epsilon_y)u_t = \mu \left((1 - \epsilon_y)\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) u + (1 - \epsilon_y) f \tag{A.20}$$

near a y boundary, and

$$(1 - \epsilon_{xy})u_t = \mu \left((1 - \epsilon_y)\frac{\partial^2}{\partial x^2} + (1 - \epsilon_x)\frac{\partial^2}{\partial y^2} \right) u + (1 - \epsilon_{xy}) f \tag{A.21}$$

near x and y boundaries, where ϵ_x and ϵ_y correspond to the error in the discrete second x and y derivatives, respectively, and $1 - \epsilon_{xy} = (1 - \epsilon_x)(1 - \epsilon_y)$. Let the subscripts x and y denote the 1-dimensional operator for the x and y dimension, respectively, of the modified identity, $\tilde{I}$, second derivative, L, and boundary operator B. We write the discretization of Equations (A.19)–(A.21) with Crank-Nicolson timestepping succinctly as

$$(\tilde{I}_y \odot \tilde{I}_x)\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\Delta t} = \frac{\mu}{2}(\tilde{I}_y \odot L_x + L_y \odot \tilde{I}_x)(\boldsymbol{u}^{n+1} + \boldsymbol{u}^n) + (\tilde{I}_y \odot \tilde{I}_x)f^{n+1/2}, \tag{A.22}$$

Letting $\tilde{I} = \tilde{I}_y \odot \tilde{I}_x$, $\nabla_h^2 = \tilde{I}_y \odot L_x + L_y \odot \tilde{I}_x$, and $B_h = \tilde{I}_y \odot B_x + B_y \odot \tilde{I}_x$, we can express Equation (A.22) identically to its one-dimensional analog, Equation (A.15). For boundaries that do not require correction, $\tilde{I} \equiv I$, and the recursion (A.16) can be used to construct even higher-dimensional operators by replacing identity operators with their boundary-corrected counterparts. The system is symmetric positive-definite if every $\epsilon > 0$, and can be solved via conjugate gradients.

References

[106] F. H. Harlow and J. E. Welch, *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, Physics of Fluids, 8 (1965), pp. 2182–2189.

[107] P. Wesseling, *Principles of computational fluid dynamics*, vol. 29 of Springer Series in Computational Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

APPENDIX B

DISCRETE GEOMETRY OF A FLAT TORUS

While our choice of force density for the endothelium does not require surface reconstruction or quadrature weights, the methods described in Section 2.3 are equally applicable. The periodicity of Ω makes the endothelium a topological torus. The 4-dimensional torus, $\mathbb{T}$, with parametrization

$$\boldsymbol{\chi}(\theta,\, \varphi) = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \cos\varphi \\ \cos\varphi \end{bmatrix},$$

for $(\theta,\, \varphi) \in [0,\, 2\pi)^2$, is orientation agnostic. Using distances between points on $\mathbb{T}$ to construct RBFs allow for the interpolation of any toroidal surface. The parametrized metric is given by

$$\begin{aligned} \|\boldsymbol{\chi}(\theta_j,\, \varphi_j) - \boldsymbol{\chi}(\theta_i,\, \varphi_i)\| &= \sqrt{2 - \cos(\theta_j - \theta_i) - \cos(\varphi_j - \varphi_i)} \\ &= \sqrt{2 - \boldsymbol{\chi}(\theta_j,\, \varphi_j) \cdot \boldsymbol{\chi}(\theta_i,\, \varphi_i)}. \end{aligned} \tag{B.1}$$

$\mathbb{T}$ is therefore homogeneous, and this parametrization can be used to compute quadrature weights. It has surface area $4\pi^2$ and Jacobian $\sqrt{\det(\mathcal{G}_0(\boldsymbol{Z}_{\mathbb{T}}))} = 1$ (see Section 3.2.4.2). It is, however, trivial to choose a set of points on $\mathbb{T}$ that generate equal quadrature weights without the need to solve a system like (2.22). Any regular grid with correct spacing across the periodic boundaries suffices, but we generate a set of N points, whether used for interpolation or not, with the spiral

$$\begin{aligned} \varphi_i &= 2\pi(i-1)/N, \\ \theta_i &= \operatorname{mod}\left(\left\lceil\sqrt{N}\right\rceil \varphi_i,\, 2\pi\right). \end{aligned}$$

However, care needs to be taken in applying the constructed discrete operators to a "flat" torus, such as the endothelium. If, for example, L is the discrete analogue of $\mathcal{L}$ evaluated at sample site $(\theta^{\mathrm{s}},\, \varphi^{\mathrm{s}})$ and the interpolated function $\psi(\theta,\, \varphi)$ decomposes into

$$\psi(\theta,\, \varphi) = \bar{\psi}(\theta,\, \varphi) + \text{periodic part},$$

where $\bar{\psi}$ is known, aperiodic, and independent of time, then

$$\mathcal{L}\psi(\theta^{\mathrm{s}},\, \varphi^{\mathrm{s}}) \approx \mathcal{L}\bar{\psi}(\theta^{\mathrm{s}},\, \varphi^{\mathrm{s}}) + L\left(\boldsymbol{\psi}^{\mathrm{d}} - \bar{\boldsymbol{\psi}}^{\mathrm{d}}\right) := L\boldsymbol{\psi}^{\mathrm{d}} + \epsilon,$$

where $\boldsymbol{\psi}^{\mathrm{d}}$ and $\bar{\boldsymbol{\psi}}^{\mathrm{d}}$ are vectors formed by evaluating ψ and $\bar{\psi}$, respectively, at each point in Θ^{d}. Computing ϵ for each sample site can be done simultaneously, resulting in the vector $\boldsymbol{\epsilon}$. A correction like $\boldsymbol{\epsilon}$ may be needed for each linear operator.