A fast, robust, and non-stiff Immersed Boundary Method

Hector D. Ceniceros

Department of Mathematics, University of California Santa Barbara, CA 93106

Jordan E. Fisher

Department of Mathematics, University of California Santa Barbara, CA 93106

Abstract

We propose a fast and non-stiff approach for the solutions of the Immersed Boundary Method, for Newtonian, incompressible flows in two or three dimensions. The proposed methodology is built on a robust semi-implicit discretization introduced by Peskin in the late 70's which is solved efficiently through the novel use of a fast, treecode strategy to compute flow-structure interactions. Optimal multipole-type expansions are performed numerically by solving a least squares problem with a new, fast iterative algorithm. The new Immersed Boundary Method is particularly well suited for three-dimensional applications and/or for problems where the number of immersed boundary points is large. We demonstrate the efficacy and superiority of the method over existing approaches with two simple but illustrative examples in 3D.

Keywords: semi-implicit method, Navier-Stokes equations, treecode, Krylov subspace method

---

1. Introduction

The Immersed Boundary (IB) Method, introduced by Peskin [?], offers a flexible approach for the simulation of flow-structure interaction. It combines

Email addresses: `hdc@math.ucsb.edu` (Hector D. Ceniceros), `www.math.ucsb.edu/~hdc` (Hector D. Ceniceros), `jordan@math.ucsb.edu` (Jordan E. Fisher) a Lagrangian representation of the immersed structures with an Eulerian flow description. The Lagrangian representation of the immersed boundaries endows the method with a versatile structure-building capability while the Eulerian flow description permits the use of fast flow solvers. The connection of the two representations is done seamlessly through *spreading* (of interfacial forces) and *interpolation* (of velocity at the immersed boundary) steps via mollified delta functions.

The immersed structures often have very stiff components and as a consequence strong *tangential* forces are generated, which in turn induce severe time-step restrictions for explicit discretization [? ?]. Fully implicit discretizations remove this hindering constraint but are seemingly impractical due to their elevated cost [? ?]. Hou and Shi [? ?] proposed recently a cost effective 2D semi-implicit method but it is applicable only to simple periodic interfaces. In [?] we presented, in collaboration with Roma, robust 2D methods for efficiently removing this stiffness for more general immersed structures, which need not be periodic or even continuous and can include cross links, tethers, etc. Indeed, it was demonstrated in [?] that the proposed approaches are applicable to a wide range of structure geometries and fiber forces and can provide a speedup of many orders of magnitude in comparison to explicit methods.

Unfortunately, the computational efficiency enjoyed by 2D methods introduced in [?] does not directly carry over to the three-dimensional case. They rely crucially on the construction of a matrix that encodes the flow-mediated interactions between immersed boundary points at each timestep and which we refer to as the *flow-structure operator*. The size of this matrix is directly related to the number of immersed boundary points. In 3D, there may be drastically more immersed boundary points, and the resulting matrix is simply too large to construct explicitly. Hence, we must turn to matrix-free alternatives.

In this paper we present one such alternative. The new method makes use of a treecode strategy and is much more robust to increases in fiber points. In fact, the efficiency is superior enough to allow the method to be used even when the number of fiber points approaches parity with that of fluid cells. Here, as it is customary in the IB Method, we consider only periodic boundary conditions. The assumption of periodic boundary conditions is critical to one aspect of the proposed method (the near translation invariance). However, Dirichlet type of boundary conditions can be easily implemented in the IB framework by treating solid boundaries as being immersed within a periodic domain.

In line with the 2D methodology in [?], our starting point is the robust, semi-implicit scheme introduced by Peskin [?] in the late 70's, in which the spreading and interpolation operators are lagged, i.e. evaluated at the current interfacial configuration rather than at the future one. Newren, Fogelson, Guy, and Kirby proved that this scheme, in its first order or second order Crank-Nicolson form, is unconditionally stable when inertia is neglected and the interfacial force is linear and self-adjoint [?]. Numerical experiments in [?], as well as our own extensive experiments, suggest the robustness of this discretization extends to the inertial case with nonlinear interfacial force.

This semi-implicit discretization leads to a system of equations, generally nonlinear, for the interface configuration at the future timestep. The dominant computational cost of solving this system is the flow-structure operator which was expressed as a matrix in the 2D methodology [?]. From this perspective, there are two interrelated problems for removing robustly the numerical stiffness of the IB Method:

1. The design of efficient solvers for the (nonlinear) system of the interface configuration.
2. The fast evaluation of the flow-structure operator.

In the current work, we focus on the second problem. We show that the flow-structure operator can be seen as a *multipole summation* with a suitable choice of potential. Using the Singular Value Decomposition and a new, efficient, iterative algorithm we compute L^2-optimal far field expansions of this potential to be used in an effective treecode strategy. This treecode approach allows for a very fast evaluation of the flow-structure operator. With that in hand, we solve the implicit system for the interface configuration with a Krylov subspace method (i.e. Problem 1), employing the treecode evaluation at every iteration.

We demonstrate here that this treecode-based Krylov subspace method yields an implicit solver that is asymptotically faster than a single fluid solve. This contrasts with the implicit solver from the 2D case, which had asymptotic cost equal to a fluid solve. Thus, we will see that the present methodology is not only an extension of the previous methodologies to the 3D case, but also a substantial improvement upon them.

The presentation is organized as follows. In Section 2, we review the formulation of the IB Method. Section 3 presents the discretization with the focus on Peskin's semi-implicit scheme with lagged-operators. Following this, Section 4 provides a brief overview of the computational difficulties associated with the IB Method, as well as detailing why the 2D methodology proposed in [?] cannot be directly extended to the 3D case. Section 5 is devoted to the treecode approach, including a sketch of how to arrive at the needed multipole expansions. A detailed consideration of the multipole expansions is presented in Appendix B. With the treecode in hand, we proceed to test problems in Section 6. Finally, some concluding remarks are provided in Section 7.

2. The Immersed Boundary Method

We now review the IB Method in its simplest form. We consider an incompressible, Newtonian fluid occupying a domain $\Omega \subset \mathbb{R}^3$. Inside this domain we assume that there is an immersed, neutrally buoyant, elastic structure. This structure may be a 1D filament or a 2D surface, but may also be a more complex, dense 3D mesh, or some combination of all these elements. The structure need not be closed or even continuous. We refer to the set of points comprising the structure as Γ. We further assume there is some parametrization of Γ, given over a parameter space B. The configuration of Γ at time t is then provided in the Lagrangian form $\mathbf{X}(s,t)$, where $s \in B$ is a Lagrangian parameter. The fluid and immersed structure form a coupled system evolving according to:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \mathbf{f}, \tag{1}$$

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

$$\frac{\partial \mathbf{X}}{\partial t} = \mathbf{u}(\mathbf{X},t), \tag{3}$$

where ρ and μ are the fluid density and fluid viscosity, respectively (both assumed to be constant). Here $\mathbf{u}(\mathbf{x},t)$ and $p(\mathbf{x},t)$ are the velocity field and the pressure, respectively, described in terms of the Eulerian, Cartesian coordinate $\mathbf{x}$. The term $\mathbf{f}$ represents the singularly supported tension force of the immersed structure acting onto the fluid. The system (1)-(3) is supplemented with initial and boundary conditions. Throughout this work, we consider only periodic boundary conditions and take $\Omega = [0,1]^3$.

The crux of the IB Method and much of its versatility is the seamless connection of the Lagrangian representation of the immersed structure with the Eulerian representation of the flow. This is achieved via the identities:

$$\frac{\partial \mathbf{X}}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\mathbf{X}(s,t))d\mathbf{x}, \tag{4}$$

$$\mathbf{f}(\mathbf{x},t) = \int_B \mathbf{F}(s,t)\delta(\mathbf{x}-\mathbf{X}(s,t))ds, \tag{5}$$

where δ denotes the (three-dimensional) Dirac delta distribution. In (5), $\mathbf{F}$ represents the elastic force density of Γ and is described in Lagrangian coordinates. Typically, the fiber force at time t, $\mathbf{F}(\cdot,t)$, is given as a function of $\mathbf{X}(\cdot,t)$, the configuration of our immersed structure at time t. We denote this as

$$\mathbf{F} = \mathcal{A}(\mathbf{X}), \tag{6}$$

for some potentially nonlinear operator $\mathcal{A}$.

3. Discretization

We discretize Ω as a uniform $N \times N \times N$ Cartesian grid $\mathcal{G}_\Omega$ with grid size $h = 1/N$. We represent our structure Γ as a collection of N_B points and define $h_B = 1/N_B$. Note that different conventions may be used to discretize the Lagrangian parameter space B when Γ is not one-dimensional. We call our discretization $\mathcal{G}_B$ and take it to be a one dimensional index space, so that the discretized structure $\mathbf{X}$ may be considered as an array of N_B 3-vectors, $\{\mathbf{X}_i\}_{i=1}^{N_B}$.

We employ a semi-implicit discretization for the equations of motion [**?**],

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} + \mathbf{u}^n \cdot \mathbf{D}_h \mathbf{u}^n\right) = -\mathbf{D}_h p^{n+1} + \mu L_h \mathbf{u}^{n+1} + \mathcal{S}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}), \tag{7}$$

$$\mathbf{D}_h \cdot \mathbf{u}^{n+1} = 0, \tag{8}$$

$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^n}{\Delta t} = \mathcal{S}_n^* \mathbf{u}^{n+1}, \tag{9}$$

where a superscript n denotes a numerical approximation taken at the time $n\Delta t$ and Δt is the timestep. We use standard, second order finite differences for the spatial derivatives. The spatial operators $\mathbf{D}_h$ and L_h are the standard, second order approximations to the gradient and the Laplacian, respectively, and $\mathcal{A}_{h_B}$ is a suitable discrete version of $\mathcal{A}$.

$\mathcal{S}_n$ and $\mathcal{S}_n^*$ are the *lagged* spreading and interpolation operators, respectively, given by

$$(\mathcal{S}_n G)(\mathbf{x}) = \sum_{s \in \mathcal{G}_B} G(s)\delta_h(\mathbf{x} - \mathbf{X}^n(s))h_B, \tag{10}$$

$$(\mathcal{S}_n^* w)(s) = \sum_{\mathbf{x} \in \mathcal{G}_\Omega} w(\mathbf{x})\delta_h(\mathbf{x} - \mathbf{X}^n(s))h^3, \tag{11}$$

where $\delta_h(\mathbf{x}) = d_h(\mathbf{x}_0)d_h(\mathbf{x}_1)d_h(\mathbf{x}_2)$ and d_h is an approximation of the one-dimensional Dirac delta distribution. These operators are called lagged because the interface configuration $\mathbf{X}^n$ is used instead of the future configuration $\mathbf{X}^{n+1}$. With this choice, $\mathcal{S}_n$ and $\mathcal{S}_n^*$ are linear with respect to the future configuration. We stress that this linearity is critical to the numerical efficiency of our algorithm.

There is flexibility in the choice of d_h but for concreteness in the presentation we choose Peskin's delta [**?**]:

$$d_h(r) = \begin{cases} \frac{1}{4h}\left(1 + \cos(\frac{\pi r}{2h})\right) & \text{if } |r| \leq 2h, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

We note that the analogous semi-implicit discretization for Stokes flow is known to be stable when $\mathcal{A}_{h_B}$ is linear and self-adjoint, as proved in [**?**]. Our own extensive experiments, as well as results presented in [**?**], suggest that (7)-(9) are also stable for convective flows with non-linear $\mathcal{A}_{h_B}$, with only a mild CFL condition.

The discretization (7)-(9) presents a formidable implicit system to solve. Before outlining a method of solution, we seek a simpler representation. Let us rewrite (7) as

$$\mathbf{u}^{n+1} = -\frac{\Delta t}{\rho}\mathbf{D}_h p^{n+1} + \nu \Delta t L_h \mathbf{u}^{n+1} + \mathbf{a}^{n+1}, \tag{13}$$

where $\nu = \mu/\rho$ and

$$\mathbf{a}^{n+1} = \frac{\Delta t}{\rho}\mathcal{S}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{u}^n - \Delta t\ \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n. \tag{14}$$

We can eliminate the pressure term in (13) using (8), by introducing a discrete projection P_h defined as

$$\mathbf{v} = P_h\mathbf{v} + \mathbf{D}_h\phi_v, \qquad \mathbf{D}_h \cdot P_h\mathbf{v} = 0, \qquad P_h\mathbf{D}_h\phi_v = 0, \tag{15}$$

for any smooth vector field $\mathbf{v}$ defined on the grid $\mathcal{G}_\Omega$. Applying P_h to (13), using (8), and the fact that for periodic boundary conditions L_h and P_h commute we get

$$\mathbf{u}^{n+1} = \nu\Delta t L_h \mathbf{u}^{n+1} + P_h \mathbf{a}^{n+1}, \tag{16}$$

that is

$$\mathbf{u}^{n+1} = (I - \nu\Delta t L_h)^{-1} P_h \mathbf{a}^{n+1}. \tag{17}$$

Let us denote

$$\mathcal{L}_h = (I - \nu\Delta t L_h)^{-1} P_h. \tag{18}$$

We refer to $\mathcal{L}_h$ as the *fluid solver*. For calculations of the form $\mathcal{L}(\mathbf{f})$ we rely on an FFT based method, although any suitable method could be employed. Using $\mathcal{L}_h$, the semi-implicit method (7)-(9) can be expressed as

$$\mathbf{u}^{n+1} = \mathcal{L}_h \mathbf{a}^{n+1}, \tag{19}$$

$$\mathbf{X}^{n+1} = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathbf{u}^{n+1}, \tag{20}$$

where $\mathbf{a}^{n+1}$ is given by (14). Eliminating $\mathbf{u}^{n+1}$ in (20) we obtain a system of equations for the immersed boundary configuration $\mathbf{X}^{n+1}$:

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{b}^n, \tag{21}$$

where

$$\mathcal{M}_n = \alpha\, \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n, \tag{22}$$

with

$$\alpha = \frac{(\Delta t)^2}{\rho} \tag{23}$$

and

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h[\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n]. \tag{24}$$

We have thus reduced (7)-(9) to a single system of equations involving only the unknown $\mathbf{X}^{n+1}$. After solving this system we can obtain $\mathbf{u}^{n+1}$ via (19). We call the linear operator $\mathcal{M}_n$ the *flow-structure operator*.

4. Building an efficient and robust, non-stiff IB Method

Within our framework, there are two main difficulties to produce a cost-efficient and robust, non-stiff IB Method. The first is the heavy cost of applying and inverting the flow-structure operator $\mathcal{M}_n$. The second is to produce an effective, in general nonlinear, iterative solver for the implicit system (21). Because $\mathcal{A}_{h_B}$ can be a very general nonlinear operator, the second problem can prove difficult.

In [**?**] we approached both of these problems. The computational cost of evaluating expressions of the form $\mathcal{M}_n\mathbf{F}$ was reduced by representing the linear operator in matrix form. We employed a multigrid to efficiently solve systems of the form $\mathcal{M}_n\mathbf{F} = \mathbf{b}$ and $(I - \mathcal{M}_n\mathcal{A}_{h_B})\mathbf{X} = \mathbf{b}$, for linear $\mathcal{A}_{h_B}$.

The second problem is more application dependent. Provided the Jacobian J of $\mathcal{A}_{h_B}$ is negative semi-definite, we showed that Newton's iterations of the system (21) converge well, as was also demonstrated by Mori and Peskin in [**?**]. The linear system obtained at each iteration of Newton's method can be solved with the same techniques used in the case where $\mathcal{A}_{h_B}$ is linear. Handling a non-definite J is more challenging. In [**?**] a splitting algorithm was proposed which performs well for the wide class of functions $\mathcal{A}_{h_B}$ such that the eigenvalues of J are all large in magnitude. Additional work remains to be done for the case where the eigenvalues of J are mixed in magnitude.

The present work focuses on the first problem: the large computational cost associated with flow-structure operator $\mathcal{M}_n$. The matrix form of $\mathcal{M}_n$ has $O(N_B^2)$ terms. In 2D, we typically have $N_B \sim N$. The cost of evaluating $\mathcal{M}_n\mathbf{F}$ is then $O(N^2)$, which is slightly smaller than the $O(N^2 \log N)$ cost of applying $\mathcal{L}_h$. In 3D, an immersed surface may require $N_B \sim N^2$ fiber points. The resulting matrix form of $\mathcal{M}_n$ would then have $O(N^4)$ terms, which is much greater than the $O(N^3 \log N)$ cost of applying $\mathcal{L}_h$. Directly extending the matrix method to 3D is therefore impractical. We instead turn to an alternative method for efficiently handling $\mathcal{M}_n$ within our implicit system.

To isolate the problem at hand we consider only simple, linear forcing functions $\mathcal{A}_{h_B}$. We may rewrite (21) as the linear system

$$(I - \mathcal{M}_n\mathcal{A}_{h_B})\mathbf{X}^{n+1} = \mathbf{b}^n, \tag{25}$$

where I is the identity operator. If $\mathcal{A}_{h_B}$ is negative semi-definite then $I - \mathcal{M}_n\mathcal{A}_{h_B}$ is positive definite. A multitude of iterative methods exist to efficiently solve positive definite systems. For this work, we make use of the matrix-free, Conjugate Gradient Method (CG). Iterations of CG require an evaluation of the operator $I - \mathcal{M}_n\mathcal{A}_{h_B}$, and hence an evaluation of $\mathcal{M}_h$. CG can require upward of a dozen iterations to achieve adequate convergence, so it is critical to streamline the evaluations of $\mathcal{M}_h$.

Our solution to this problem is to adapt a treecode strategy for use with the IB Method. The methodology is elaborated in the following section.

While we focus here on a simple, linear $\mathcal{A}_{h_B}$, the proposed methodology can also be applied directly to more general linear $\mathcal{A}_{h_B}$ as long as they are negative semi-definite. Moreover, it is also applicable when $\mathcal{A}_{h_B}$ is nonlinear but has a negative semi-definite Jacobian. In such nonlinear situations, as demonstrated in 2D [**?**], Newton's method in concert with a fast methodology to evaluate flow-structure interactions, yields an efficient method. As mentioned above, fiber forces with a non-definite Jacobian with both large and small eigenvalues are more challenging and remain to be investigated.

5. Treecode Evaluation

Treecodes are efficient ways to evaluate certain sums. Suppose we have an expression of the form

$$\sum_{j=1}^{N_B} \phi(\mathbf{X}_j, \mathbf{X}_i)\mathbf{F}_j, \tag{26}$$

where $\{\mathbf{F}_j\}_{j=1}^{N_B}$ is a collection of forces, and ϕ is some tensor valued potential. Expression (26) is referred to as a multipole summation. Evaluating (26) for all $1 \leq i \leq N_B$ directly requires $O(N_B^2)$ operations. However, treecodes can reduce the overall computational cost to $O(N_B \log N_B)$, provided ϕ is sufficiently regular.

To make use of a treecode approach within the IB Method context, we must first recast products of the form $\mathcal{M}_n\mathbf{F}$ as multipole summations. Recall $\mathcal{M}_n = \alpha\, \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n$ is a linear function from $\mathbb{R}^{3N_B}$ to $\mathbb{R}^{3N_B}$. For a componentwise calculation of $\mathcal{M}_n$, we focus on two fiber points located at $\mathbf{x} \in \Omega$ and $\mathbf{y} \in \Omega$. We take $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ to be the canonical basis vectors of $\mathbb{R}^3$ and define the scalar field

$$\delta_{\mathbf{x}}(\mathbf{z}) \equiv \delta_h(\mathbf{z} - \mathbf{x}). \tag{27}$$

Hence, $\mathbf{e}_i\delta_{\mathbf{x}}$ is the field obtained by spreading the unit force $\mathbf{e}_i$ at $\mathbf{x}$ to the fluid domain. We denote the influence of this force field on the fluid velocity as $\mathbf{u}_{\mathbf{x}}^i \equiv \mathcal{L}_h\mathbf{e}_i\delta_{\mathbf{x}}$. The induced velocity on $\mathbf{y}$ is now just an interpolation of $\mathbf{u}_{\mathbf{x}}^i$ at $\mathbf{y}$. We perform this procedure for $i = 1, 2, 3$ and store the three resulting vectors as a 3×3 matrix (tensor) defined via

$$(G_h(\mathbf{x}, \mathbf{y}))_{ij} \equiv \alpha h_B \sum_{\mathbf{z}\in\mathcal{G}_\Omega} u_{\mathbf{x}j}^i(\mathbf{z})\delta_h\left(\mathbf{z} - \mathbf{y}\right) h^3, \tag{28}$$

where $u^i_{\mathbf{x}j}$ is the j-th component of $\mathbf{u}^i_{\mathbf{x}}$ and the summation against δ_h provides the necessary interpolation. G_h is a tensor valued function, acting on $\Omega \times \Omega$ and returning a 3×3 matrix that specifies how forces at $\mathbf{x}$ affect the fiber displacement at $\mathbf{y}$. If there is a force $\mathbf{f}$ on $\mathbf{x}$ then the induced displacement at $\mathbf{y}$ is simply $G_h(\mathbf{x}, \mathbf{y})\mathbf{f}$. Evaluating G_h for all ordered pairs $(\mathbf{X}_i, \mathbf{X}_j)$, we have that

$$(\mathcal{M}_n \mathbf{F})_i = \sum_{0 \leq j \leq N_B} G_h(\mathbf{X}_j, \mathbf{X}_i)\mathbf{F}_j, \qquad \text{for } 0 \leq i \leq N_B, \tag{29}$$

which is exactly of the form (26). One can also arrive at this representation by making use of the Fourier transform.

In practice, calculating $G_h(\mathbf{x}, \mathbf{y})$ for any pair of points $\mathbf{x}, \mathbf{y} \in \Omega$ is prohibitively expensive. It would require a fluid solve, i.e. $O(N^3 \log N)$ operations per pair of points. To overcome this limitation, following [**?**], we approximate $G_h(\mathbf{x}, \mathbf{y})$ by $G_h(\mathbf{0}, \mathbf{x}-\mathbf{y})$, which we denote simply as $G_h(\mathbf{x}-\mathbf{y})$. G_h can now be seen as a tensor field over Ω. We precompute and store the values of $G_h(\mathbf{z})$ for every $\mathbf{z} \in \mathcal{G}_\Omega$. This allows us to reduce future costs of evaluating $G_h(\mathbf{z})$ to $O(1)$. When $\mathbf{z} \notin \mathcal{G}_\Omega$ we use trilinear interpolation to approximate $G_h(\mathbf{z})$.

For a given set of parameters $(\Delta t, \mu, \rho, h, \Omega)$ defining the discretized fluid domain and the fluid material properties, G_h need only be calculated once and can be reused throughout a simulation and in fact, in any other simulations with identical fluid parameters. The price we pay for this computational speedup is the error incurred by assuming translational invariance, as well as by using interpolation between grid points. As we show below, these errors do not degrade the overall accuracy of the IB Method.

In the continuous case the flow-mediated interaction between two fiber points is (exactly) translation invariant, that is,

$$\lim_{h \to 0} G_h(\mathbf{x}+\mathbf{z}, \mathbf{y}+\mathbf{z}) = \lim_{h \to 0} G_h(\mathbf{x}, \mathbf{y}), \qquad \text{for any } \mathbf{z} \in \Omega. \tag{30}$$

Thus, in the limit as $h \to 0$, the approximation $G_h(\mathbf{x}, \mathbf{y}) = G_h(\mathbf{x}-\mathbf{y})$ is exact. In the case of a 2D fluid, it was proved in [**?**] that each component of the error $|(G_h(\mathbf{x}, \mathbf{y}) - G_h(\mathbf{x}-\mathbf{y}))_{ij}|$ is smaller than $O(h)$ and does not deteriorate the global accuracy of the IB Method. In 3D, the same argument carries through with minimal modification (see Appendix C).

We note that translation invariance here is dependent on the type of boundary conditions on our domain Ω. In particular, Ω must be either periodic or infinite ($\Omega = \mathbb{R}^3$). This requirement is not as restrictive as it might seem as Dirichlet type of boundary conditions can be easily implemented in the IB framework as immersed boundaries within a periodic domain.

Given our approximation to G_h we can now cast our fluid evaluations as multipole summations of the form

$$(\mathcal{M}_n \mathbf{F})_i = \sum_{0 \leq j \leq N_B} G_h(\mathbf{X}_j - \mathbf{X}_i)\mathbf{F}_j. \tag{31}$$

Calculating this sum directly is equivalent to the matrix method proposed in [**?**], and requires $O(N^4)$ operations when $N_B \sim N^2$. The goal now is to apply a treecode algorithm to accelerate the evaluation of this sum, ideally reducing the cost from $O(N_B^2)$ to $O(N_B \log N_B)$. In the following two subsections we overview the basics of treecodes. We will elaborate in Section 5.3 on the far field expansions of G_h that enable a treecode strategy to perform well for our particular problem.

5.1. Overview

For those unfamiliar with treecodes we briefly review the main ideas here. We recommend an alternative exposition by Li, Johnston, and Krasny in [**?**].

The general strategy is to make use of far field expansions of G_h to compress the outgoing effect of clusters of fiber points. Assume we have two subsets of our domain Ω, Ω_{in} and Ω_{out}, such that G_h has a valid expansion in $\Omega_{in} \times \Omega_{out}$ given by

$$G_h(\mathbf{x} - \mathbf{y}) \approx \sum_{k=1}^{p} A_k(\mathbf{x})B_k(\mathbf{y}), \qquad \text{for } \mathbf{x} \in \Omega_{in} \text{ and } \mathbf{y} \in \Omega_{out}. \tag{32}$$

Ω_{out} serves to restrict the location of our pole, the *outgoing* influence, while Ω_{in} serves to restrict the point of evaluation, the *incoming* position. Our expansion terms $\{A_k\}_{k=1}^p$ and $\{B_k\}_{k=1}^p$ are collections of 3×3 matrix (tensor) fields defined on Ω_{in} and Ω_{out} respectively, with multiplication between two tensors defined componentwise, and multiplication between a tensor and a vector defined via the usual matrix-vector multiplication. If we further define

$$B_{in} = \{1 \leq i \leq N_B | \mathbf{X}_i \in \Omega_{in}\}, \tag{33}$$

$$B_{out} = \{1 \leq i \leq N_B | \mathbf{X}_i \in \Omega_{out}\}, \tag{34}$$

then we can consider the subproblem of calculating the influence of all the fiber points in Ω_{out} on the fiber points in Ω_{in}:

$$\sum_{j \in B_{out}} G_h(\mathbf{X}_i - \mathbf{X}_j)\mathbf{F}_j, \qquad \text{for } i \in B_{in}. \tag{35}$$

Calculating this summation for all i in B_{in} requires $O(|B_{in}|\cdot|B_{out}|)$ operations, where $|\cdot|$ denotes the number of elements in a set. We seek to reduce this cost. Substituting (32) in (35) yields

$$\begin{aligned}\sum_{j \in B_{out}} G_h(\mathbf{X}_j - \mathbf{X}_i)\mathbf{F}_j &\approx \sum_{j \in B_{out}} \left(\sum_{k=1}^{p} A_k(\mathbf{X}_i) B_k(\mathbf{X}_j) \right) \mathbf{F}_j \\ &= \mathbf{E}^T \sum_{k=1}^{p} A_k(\mathbf{X}_i) \sum_{j \in B_{out}} B_k(\mathbf{X}_j) \tilde{F}_j. \end{aligned} \tag{36}$$

Here $\tilde{F}_j$ is a 3×3 matrix (tensor) where $(\tilde{F}_j)_{ab} = (\mathbf{F}_j)_b$ and $\mathbf{E}$ is a 3-vector whose components are all one. The additional complication of defining $\tilde{F}_j$ arises because the product of A_k and B_k in (32) is understood component-wise. The vector $\mathbf{E}$ serves to collapse the final sum into a vector.

The summation over B_{out} in (36) may be calculated independently of i. Thus, given

$$H_k \equiv \sum_{j \in B_{out}} B_k(\mathbf{X}_j) \tilde{F}_j, \tag{37}$$

we can efficiently calculate (35) via

$$\mathbf{E}^T \sum_{k=1}^{p} A_k(\mathbf{X}_i) H_k, \qquad \text{for } i \in B_{in}. \tag{38}$$

Using (38) to compute (35) requires $O(p|B_{in}| + |B_{out}|)$ operations, which may be substantially fewer than $O(|B_{in}| \cdot |B_{out}|)$ if $p \ll |B_{out}|$. This compression is one of the main ingredients of a treecode. The remaining difficulty is that Ω_{in} and Ω_{out} generally do not cover our entire domain. There may be fiber points $\mathbf{X}_i$ and $\mathbf{X}_j$ where i, j do not belong to B_{out}, B_{in}, hence the interaction between $\mathbf{X}_i$ and $\mathbf{X}_j$ would not be accounted for in (35).

To remedy this we must choose multiple pairs of $(\Omega_{in}, \Omega_{out})$ so that every ordered pair of fiber points $(\mathbf{X}_i, \mathbf{X}_j)$ is represented exactly once. This is essentially an organizational problem, and the standard procedure is to use binary space partitioning. In 3D this is known as an octree whereas in 2D it is called a quadtree.

5.2. The octree

An octree is constructed by successively subdividing our domain $\Omega = [0,1]^3$ into smaller cubic domains called *panels*. Starting with the domain itself, called the *Root Panel*, we divide it into eight equal octants called child panels (of the root) then recursively we define the eight children of each of those child panels and so on. This subdivision process is stopped for a panel which has fewer that an (arbitrarily set) minimum number of fiber points (10 here).

The totality of the Root Panel and all of its branches is collectively known as the octree. Visualizing an octree in 3D is difficult. For demonstrative purposes we present in Figure 1 a drawing of a quadtree in 2D.

Given a panel P we refer to its domain as Ω^P_{out}. A point $\mathbf{x}$ is *well-separated* from a panel P if its distance to the center of the panel is at least

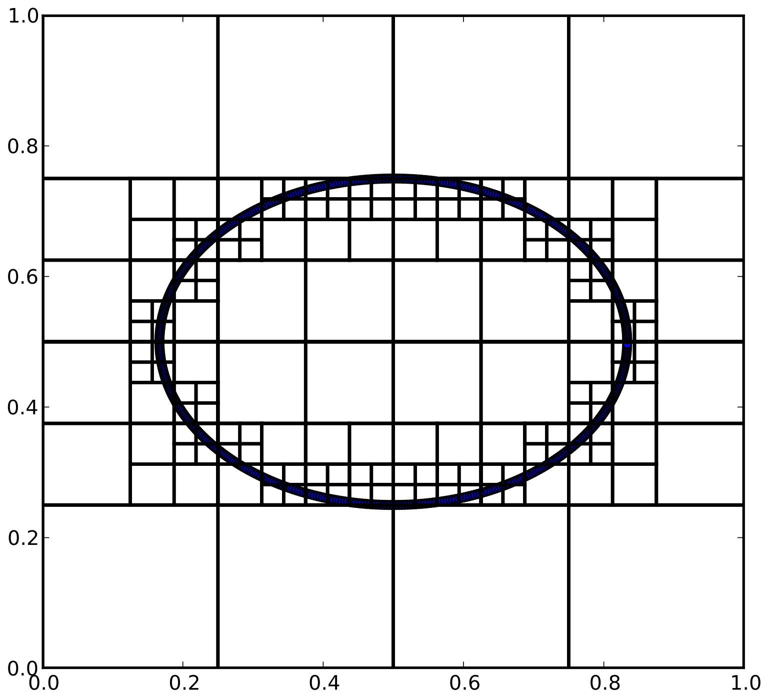


Figure 1: An ellipse shaped fiber $\mathbf{X}$, with a quadtree decomposition of the space $[0,1] \times [0,1]$ containing it.

3/2 the size of the panel [see (A.2)]. We refer to the domain of all points well-separated from P as Ω^P_{in}. We assume that points in Ω^P_{in} are sufficiently far from P that a multipole expansion could be used to calculate the effect of the *outgoing* influence of all the fiber points in P. This assumption is equivalent to assuming that we can expand G_h over $\Omega^P_{in} \times \Omega^P_{out}$, arriving at two collections of coefficient functions $\{A^P_k\}^p_{k=1}$ and $\{B^P_k\}^p_{k=1}$.

Suppose now that we are given a fiber force $\mathbf{F}$ and wish to evaluate its influence at points $\mathbf{x}$ in the fluid domain. For each $\mathbf{x}$ this influence is given by

$$\sum_{1\leq j\leq N_B} G_h(\mathbf{x}-\mathbf{X}_j)\mathbf{F}_j. \tag{39}$$

To evaluate this efficiently, we first loop over each panel P and calculate the far field expansion of all the poles located in P. The k-th term of this expansion is given by

$$H^P_k \equiv \sum_{j\in B^P_{out}} B^P_k(\mathbf{X}_j)\tilde{F}_j. \tag{40}$$

If $\mathbf{x}$ is well separated from P, then the incoming effect on $\mathbf{x}$ from P is given by

$$\mathbf{E}^T \sum_{k=1}^{p} A^P_k(\mathbf{X}_i)H^P_k. \tag{41}$$

We can now evaluate the entire influence of $\mathbf{F}$ on a point $\mathbf{x}$, recursively. We start with the panel $P = \mathit{Root}$. If $\mathbf{x}$ is well separated from P then we evaluate the influence of P on $\mathbf{x}$ via (41). Otherwise, if P has children we recursively apply this process to those child panels. If instead P is childless, we directly evaluate the influence of every fiber point in P on $\mathbf{x}$, using (35). Summing all these influences from every branch of the recursion provides the desired total influence on $\mathbf{x}$.

Note that (40) only needs to be calculated once per panel, and may be reused for calculating the influence at multiple points $\mathbf{x}$. Because each fiber point $\mathbf{X}_i$ is contained in at most $\log N$ panels, the total cost of computing H^P_k for all panels P is at most $O(N_B \log N_B)$. Calculating the influence at a given $\mathbf{x}$ involves at most $O(\log N)$ panels, and hence the cost is at most $O(p \log N)$. The total cost of evaluating $\mathcal{M}_n\mathbf{F}$ is thus $O(pN_B \log N_B)$.

A detailed description of the octree creation and point evaluation is presented in Appendix A.

5.3. Expansions

In order to make use of a treecode we must be able to find expansions (32) of G_h associated with particular pairs $(\Omega_{in}, \Omega_{out})$. Here, G_h is a summation of discrete stokeslets. In free space, the continuous stokeslet has convenient analytic expansions useful for fast summation, see for instance [**?**]. The same is not true for the discrete stokeslet corresponding to a periodic domain, and much less so for the particular summation of stokeslets that yields G_h. Fortunately, an analytic expansion is not indispensable in practice. What we seek are collections of tensor valued function $\{A_k\}_{k=1}^{\infty}$ and $\{B_k\}_{k=1}^{\infty}$ defined on Ω such that

$$G_h(\mathbf{x}-\mathbf{y}) = \sum_{k=1}^{\infty} A_k(\mathbf{x})B_k(\mathbf{y}), \tag{42}$$

and, moreover, such that truncating the above expansion to p terms yields an adequate approximation, provided that $\mathbf{x}$ and $\mathbf{y}$ are well separated in some sense. That is, given two disjoint subsets Ω_{in} and Ω_{out} of our domain, we hope that for some small value of p and for a specified tolerance ϵ that

$$\left\| \left(G_h(\mathbf{x}-\mathbf{y}) - \sum_{k=1}^{p} A_k(\mathbf{x})B_k(\mathbf{y}) \right) \mathbf{f} \right\|_2 < \epsilon \left\|\mathbf{f}\right\|_2, \tag{43}$$

for all $\mathbf{f} \in \mathbb{R}^3$, $\mathbf{x} \in \Omega_{in}$, and $\mathbf{y} \in \Omega_{out}$.

For a given p, finding the optimal expansion that allows for the smallest ϵ that satisfies (43) for all $\mathbf{f}, \mathbf{x}, \mathbf{y}$ is an open question, and is likely computationally intractable.

We solve instead a least squares problem. We will search for the individual components of our tensors separately. Looking at each ab components, for $a, b = x, y, z$, we seek the expansion coefficients $\{(A_k)_{ab}\}_{k=1}^{p}$ and $\{(B_k)_{ab}\}_{k=1}^{p}$ such that we minimize the L^2-norm of the difference between G_h and our approximate expansion. That is, we seek to minimize

$$\left\| \left(G_h - \sum_{k=1}^{p} A_k B_k \right)_{ab} \right\|_2^2 \equiv \iint \left(G_h(\mathbf{x}-\mathbf{y}) - \sum_{k=1}^{p} A_k(\mathbf{x})B_k(\mathbf{y}) \right)_{ab}^2 d\mathbf{x}d\mathbf{y}, \tag{44}$$

where $A_k B_k$ is understood to be a function over $\Omega_{in} \times \Omega_{out}$, and the integrals are taken over $\mathbf{x} \in \Omega_{in}$ and $\mathbf{y} \in \Omega_{out}$. We approximate the integral in (44) as

$$\sum \left(G_h(\mathbf{x}-\mathbf{y}) - \sum_{k=1}^{p} A_k(\mathbf{x})B_k(\mathbf{y}) \right)_{ab}^2 h^6, \tag{45}$$

where the outer sum is taken over the Eulerian grid points in our subsets, $\mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h$ and $\mathbf{y} \in \Omega_{out} \cap \mathcal{G}_h$. Note that this minimization problem is decoupled with respect to the components of our tensors. We may thus consider the task of minimizing (45) as nine separate minimization problems.

It is useful to view our problem as a statement about matrices. To do this we first write $(G_h)_{ab}$ as an $N^3 \times N^3$ matrix, which we refer to as $\tilde{G}$ and which is given by the relation

$$\tilde{G}_{ij} = G_h((j_0, j_1, j_2)h - (i_0, i_1, i_2)h), \tag{46}$$

where $i = i_0 + i_1 N + i_2 N^2$ and $j = j_0 + j_1 N + j_2 N^2$ for any $0 \leq i_l < N$, $0 \leq j_l < N$, with $l = 0, 1, 2$. We may likewise write $(A_k)_{ab}$ and $(B_k)_{ab}$ as N^3-vectors labeled $\tilde{A}_k$ and $\tilde{B}_k$. We now stitch our collections of vectors $\{\tilde{A}_k\}$ and $\{\tilde{B}_k\}$ into matrices. Given p, we define an $N^3 \times p$ matrix U, a $p \times N^3$ matrix V, and a $p \times p$ diagonal matrix Σ via

$$U_{jk} = \frac{(\tilde{A}_k)_j}{\left\|\tilde{A}_k\right\|_2}, \qquad V_{kj} = \frac{(\tilde{B}_k)_j}{\left\|\tilde{B}_k\right\|_2}, \qquad \Sigma_{kk} = \left\|\tilde{A}_k\right\|_2 \left\|\tilde{B}_k\right\|_2, \tag{47}$$

for $1 \leq j \leq N^3$ and $1 \leq k \leq p$.

The p columns of U are simply the normalized vector encodings of the p coefficient functions $\{A_k\}_{k=1}^p$. The rows of V are likewise formed from $\{B_k\}_{k=1}^p$. Note that we may reorder our indices such that $\{\Sigma_{kk}\}_{k=1}^p$ is a decreasing sequence.

We can now express our sum (45) as $\left\|\tilde{G} - U\Sigma V\right\|_F^2 h^6$, where $\|\cdot\|_F$ is the Frobenius matrix norm. Minimizing the Frobenius norm here is a well-studied least squares problem. It is known that if we take $p = N^3$ and find the optimal expansion minimizing (45) then we recover $\tilde{G}$ exactly. That is, $\tilde{G} = U\Sigma V$. This is precisely the Singular Value Decomposition (SVD) of $\tilde{G}$.

The key property of this decomposition for our needs is that, for a given $p < N^3$, truncating the SVD to p terms provides the minimizing expansion for (45). That is, given the SVD of $\tilde{G}$, we can obtain the optimal p-term expansion coefficient functions $\{A_k\}_{k=1}^p, \{B_k\}_{k=1}^p$ from the first p columns of $U\sqrt{\Sigma}$ and the first p rows of $\sqrt{\Sigma}V$, respectively. This result is known as the Eckart-Young theorem [**?**] and the resulting expansion is the so-called rank-1 decomposition.

Using this optimal p-term expansion, the L^2-error (45) is given by the square sum of the neglected singular values,

$$\left(\sum_{k=p+1}^{N^3} \Sigma_{kk}^2 \right)^{1/2} . \tag{48}$$

The accuracy of our p-term expansion is thus directly related to the rate of decay of the singular values of $\tilde{G}$. The faster the singular values decay the fewer terms we require in our expansion to accurately capture the behavior of $\tilde{G}$. Figure 2 shows the sharp decay of the singular values of $\tilde{G}$, where we have encoded in $\tilde{G}$ the xx and xy components of G_h respectively. It is this marked decay that allows for an efficient treecode approach.

Calculating the SVD of $\tilde{G}$ can be expensive. Fortunately, as with the construction of G_h, this is a one time cost that can be spread over multiple simulations. However, some care must still be taken as a direct approach to calculating the SVD of $\tilde{G}$ would require $O(N^9)$ operations. In Appendix B we propose an efficient strategy to reduce the cost to $O(pN^3 \log N)$ operations.

Finally, turning back to our original optimization problem of minimizing (44), we approximate $A_k(\mathbf{x})$ and $B_k(\mathbf{x})$ at arbitrary positions $\mathbf{x} \in \Omega_{in}$ and $\mathbf{y} \in \Omega_{out}$ by using trilinear interpolation between the surrounding Eulerian grid points. G_h is smooth away from the origin, and its singular vectors are likewise smooth. Trilinear interpolation thus introduces an error of at most $O(h^2)$. For a given interaction between two fiber points we commit this error p times, for a total error on the order of $O(ph^2)$. Assuming h is sufficiently small, this error is smaller than the $O(h)$ error of our expansion and the $O(h)$ error of the IB Method. A numerical verification of the error rate of our expansion is presented below in Section 5.5.

5.4. Decomposition group

We have seen how to arrive at a decomposition of G_h over a subdomain $\Omega_{in} \times \Omega_{out}$. However, our treecode makes use of $O(N \log N)$ different subdomains. Each subdomain will require a suitable decomposition. Fortunately, we may recycle many of these decompositions for use over multiple subdomains.

Specifically, given two panels P^1 and P^2 at the same depth in our octree, we may take their expansion coefficient functions to be identical up to translation. That is, given $\mathbf{z} \in \Omega$,

$$A_k^{P_1}(\mathbf{c}^{P_1} + \mathbf{z}) = A_k^{P_2}(\mathbf{c}^{P_2} + \mathbf{z}), \tag{49}$$

$$B_k^{P_1}(\mathbf{c}^{P_1} + \mathbf{z}) = B_k^{P_2}(\mathbf{c}^{P_2} + \mathbf{z}), \tag{50}$$

provided the functions are defined at the points of evaluation. Here, $\mathbf{c}^{P_i}$ denotes the center of panel P_i, $i = 1, 2$.

Due to this equivalence we only need $\log N$ decompositions: one for each level of our octree. The total precomputational cost of calculating our lookup tables is thus $O(pN^3 \log^2 N)$, and the total memory requirement is $O(pN^3 \log N)$.

5.5. Treecode benchmarks

We present a numerical validation and performance evaluation of the treecode. To this end, we use a test problem which consists of an immersed, flat plate with fiber points $\mathbf{X}_i$ tethered to fixed points $\mathbf{T}$ and with a fiber force $\mathcal{A}_{h_B}(\mathbf{X}) = \sigma(\mathbf{T} - \mathbf{X})$ (see Section 6 for details on this problem setup). We fix $N = 128$, $\sigma = 10^7$, $\Delta t = 0.002$, and vary N_B and p.

The performance of the treecode is analyzed through the performance of two different function calls: the evaluation function, which takes in a fiber force $\mathbf{F}$ and returns the influence at every fiber point $\mathbf{X}$; and a pre-evaluation function, which does a variety of computations used to streamline the evaluation function. The pre-evaluation function is called only once per timestep

We examine accuracy first. There are two confounding factors that degrade the accuracy of our treecode. The first is that we approximate G_h as a truncated rank-1 decomposition. The second is that we introduce error into the simulation by assuming translation invariance of G_h. In order to analyze the first component separately we consider an immersed plate where the fiber points lie exactly on Eulerian intersection. This avoids the introduction of any error from translation.

We calculate the error according to the following procedure. We first make an $O(h)$ perturbation of the fiber $\mathbf{X}$. This perturbation generates a force $\mathbf{F}$ on the fiber. We then evaluate $\mathcal{M}_n\mathbf{F}$ using both the treecode and a direct fluid solve and take the sup norm of the difference.

The blue line in Figure 3 shows the resulting decrease in error for increasing values of p. If we do not restrict the fiber points to Eulerian intersections then the above procedure returns a different decay line, seen as the green line in Figure 3. Note that past $p = 10$ no additional reduction in error is achieved. This is the point at which the $O(h)$ error from our approximation to G_h overwhelms the error introduced by using a truncated expansion. In actual simulations fiber points can not be restricted to Eulerian intersections, hence any value of $p > 10$ would be computationally wasteful.

If we fix $p = 10$ and compute the error for various values of h we see that it does indeed decrease at least as fast as $O(h)$, as seen in Figure 4, consistent with the error bound presented in Appendix C.

We now analyze the performance of the treecode. All units of time are given as multiples of the average time to perform a fluid solve with $N = 128$. We first fix $p = 10$ and vary N_B. The resulting CPU time for the evaluation of $\mathcal{M}_n\mathbf{F}$ for an arbitrary $\mathbf{F}$ is shown in Figure 5. The corresponding CPU time for the pre-evaluation call is given in Figure 6. Both scale nearly linearly in N_B, as expected.

Most analytic expansions used for treecodes and FMM codes, including Taylor series expansions, are not optimal, in the sense that a different expansion would yield higher accuracy for the same number of terms. We approximate directly the L_2-optimal expansion. We note that the extension of treecodes and FMM codes to generic kernels, and the associated use of the SVD decomposition in particular, has been studied since the early 90s, see e.g. [**?**] for a more detailed exposition.

6. Numerical Results

6.1. An immersed plate.

For the first test of our proposed methodology we simulate flow past an immersed plate. The plate is a flat square of dimensions 1/2 by 1/2 and is discretized as an $(M+1)\times(M+1)$ grid of fiber points, where $M = \lfloor N\sqrt{2}/2 \rfloor$. This yields $N_B = 529$, 2116, and 8281 for $N = 32$, 64, and 128 respectively.

For each j, k such that $0 \leq j \leq M, 0 \leq k \leq M$ we have a fiber point $\mathbf{X}_{j,k} \in \Omega$. Each fiber point $\mathbf{X}_{j,k}$ is tethered to a corresponding fixed point $\mathbf{T}_{j,k} \in \Omega$ given by

$$\mathbf{T}_{j,k} = (0.25, 0.25, 0.5) + \frac{1}{2M}(j, k, 0). \tag{51}$$

The tethers induce a force $\mathcal{A}_{h_B}(\mathbf{X}) = \sigma(\mathbf{T} - \mathbf{X})$ which acts to restore the plate to equilibrium, as well as to bind the plate against the fluid flow pushing against it. The initial configuration of the plate is taken to be the equilibrium state $\mathbf{X}^0 = \mathbf{T}$. Note that, among non-empty Eulerian grid cells, the average number of fiber points per cell is roughly 2. This is the minimum density needed to prevent unreasonable spurious currents across the plate.

A fluid flow is induced by adding a time dependent forcing vector $\mathbf{f}(t)$ to the right hand side of (7). We take $\mathbf{f}(t) = 100(0, \sin\theta(t), \cos\theta(t))$, where $\theta(t) = 4\cos(6\pi t/T)/\pi$ and $T = 0.25$ is the total simulation time. The addition of $\mathbf{f}(t)$ alters the explicit term $\mathbf{b}^n$ in our implicit system to

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h[\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n + \Delta t \mathbf{f}(t^n)]. \tag{52}$$

The induced flow has a Reynolds number of about 10.

$\mathcal{A}_{h_B}$ is affine, not linear, thus we cannot directly apply CG to solve the implicit system. Suppose we denote our initial guess for $\mathbf{X}^{n+1}$ as $\mathbf{X}^{n+1,0}$. We define $\bar{\mathbf{X}}^{n+1} = \mathbf{X}^{n+1} - \mathbf{X}^{n+1,0}$ and recast our implicit system in terms of $\bar{\mathbf{X}}^{n+1}$. We define $\mathcal{A}'_{h_B}(\mathbf{X}) = -\sigma \mathbf{X}$ and a new explicit term

$$\bar{\mathbf{b}}^n = \mathbf{X}^n - \mathbf{X}^{n+1,0} + \Delta t \mathcal{S}_n^* \mathcal{L}_h[\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n + \mathcal{A}_{h_B}(\mathbf{X}^{n+1,0}) + \Delta t \mathbf{f}(t^n)]. \tag{53}$$

Our implicit system then becomes

$$\bar{\mathbf{X}}^{n+1} = \mathcal{M}_n \mathcal{A}'_{h_B}(\bar{\mathbf{X}}^{n+1}) + \bar{\mathbf{b}}^n. \tag{54}$$

Here $\mathcal{A}'_{h_B}$ is linear negative definite, thus $I - \mathcal{M}_n \mathcal{A}'_{h_B}$ is positive definite and we can solve (54) via CG. For the CG we use a convergence tolerance of 0.0001, which is at least 10 times smaller than the error of the method. For an initial guess we take $\mathbf{X}^{n+1,0} = \mathbf{X}^n$.

We perform both explicit and implicit simulations for $N = 32, 64, 128$ and various values of σ. The explicit simulation uses a standard Forward Euler/Backward Euler (FE/BE) discretization, i.e. implicit in the viscous term but explicit in all the other terms, including the tension force. The largest stable timestep is given by the empirical formula $\Delta t = 10h\sigma^{-1/2}$. Table 1 provides a list of the values of Δt required for a stable simulation using the FE/BE discretization. Note that even for modest resolutions the timestep is prohibitively small.

In marked contrast, for the semi-implicit, lagged operators discretization a constant timesteps of $\Delta t = 0.002$ is sufficient to maintain both stability and accuracy for all resolutions and values of σ. Our proposed fast solution strategy yields total computation times that are several orders of magnitude smaller than those for the popular FE/BE method. The total CPU time is

Table 1: Δt in the explicit simulations of the immersed plate for various values of N and σ. Δt is approximately the largest *stable* timestep, given by the formula $\Delta t = 10h\sigma^{-1/2}$.

	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N = 32$	$3.125\cdot10^{-5}$	$9.882\cdot10^{-6}$	$3.125\cdot10^{-6}$	$9.882\cdot10^{-7}$	$3.125\cdot10^{-7}$
$N = 64$	$1.563\cdot10^{-5}$	$4.941\cdot10^{-6}$	$1.563\cdot10^{-6}$	$4.941\cdot10^{-7}$	$1.563\cdot10^{-7}$
$N = 128$	$7.813\cdot10^{-6}$	$2.471\cdot10^{-6}$	$7.813\cdot10^{-7}$	$2.471\cdot10^{-7}$	$7.813\cdot10^{-8}$

shown in Table 2. The savings are striking; computations that would take over a month to perform with the FE/BE (even with a modest $N = 128$) can be done in minutes using the proposed new approach. Note also that as σ increases the total CPU time using the new methodology is almost invariant, whereas for the explicit FE/BE simulations the total CPU time grows markedly.

In Table 3 we provide a break down of the computational costs associated with a single implicit timestep for the case of the immersed plate. All units of time are given as multiples of the average time to perform a fluid solve for the given value of N. Of note is that the cost of performing Conjugate Gradient, typically requiring 5 to 10 iterations, is only a small contribution to the overall computational cost. The predominant costs come from the fluid solves and the treecode initialization and pre-evaluation.

Figure 7 presents a depiction of the flow using streamlines in a sequence of snapshots. A cross-section of the z-component of the velocity is also plotted below the plate. The flow has the expected periodic behavior while the structure of the plate is maintained throughout the simulation.

6.2. An oscillating spheroid

In the immersed plate test described above, the geometry of the structure is trivial and the deformations are negligible. Of course, the true power of the IB methodology lies in its seamless handling of both rigid *and* dynamic, flexible interfaces and in its structure-building capability. As an example of a simulation with a dynamic, flexible interface we consider now an immersed, oscillating spheroid given by

$$\frac{(x-0.5)^2}{(0.2+0.05\sin\theta(t))^2} + \frac{(y-0.5)^2}{(0.2+0.05\sin\theta(t))^2} + \frac{(z-0.5-0.05\cos\theta(t))^2}{(0.2-0.1\sin\theta(t))^2} = 1, \tag{55}$$

Table 2: Total CPU time in hours for the explicit and implicit simulations of the immersed plate, with varying values of σ. * denotes an extrapolated value.

		$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N =$ 32	Implicit	0.012	0.011	0.011	0.013	0.012
	Explicit	0.214	0.675	2.146	6.806	21.296

		$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N =$ 64	Implicit	0.109	0.109	0.106	0.111	0.108
	Explicit	4.072	12.906	40.813*	129.063*	408.133*

		$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N =$ 128	Implicit	0.896	0.889	0.897	0.892	0.896
	Explicit	64.779*	204.8481*	647.787*	2048.481*	6477.867*

Table 3: A break down of average CPU time for different components of the implicit timestep. Time is given as multiples of the average time to perform a fluid solve. Included in the Tree Initialization is the pre-evaluation cost.

	$N = 32$	$N = 64$	$N = 128$
Fluid Solves	2.000	2.000	2.000
Tree Initialization	1.166	1.125	1.107
Conjugate Gradient	0.394	0.327	0.295
Total	3.560	3.452	3.403

Table 4: Δt in the FE/BE explicit simulations of the immersed spheroid for various values of N and σ. Δt is approximately the largest stable timestep, given by the formula $\Delta t = 0.5 \cdot 10^{-1} h \sigma^{-1/2}$.

	$\sigma = 10^5$	$\sigma = 10^6$	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$
$N = 32$	$4.941{\cdot}10^{-6}$	$1.563{\cdot}10^{-6}$	$4.941{\cdot}10^{-7}$	$1.563{\cdot}10^{-7}$	$4.941{\cdot}10^{-8}$
$N = 64$	$2.471{\cdot}10^{-6}$	$7.813{\cdot}10^{-7}$	$2.471{\cdot}10^{-7}$	$7.813{\cdot}10^{-8}$	$2.471{\cdot}10^{-8}$
$N = 128$	$1.235{\cdot}10^{-6}$	$3.906{\cdot}10^{-7}$	$1.235{\cdot}10^{-7}$	$3.906{\cdot}10^{-8}$	$1.235{\cdot}10^{-8}$

where $\theta(t) = 2\pi t/T$, t is the current simulation time and $T = 0.25$ is the total simulation time. Equation (55) yields a spheroid centered at $(0.5, 0.5, 0.5 + 0.05\cos\theta)$ with an equatorial radius of $0.2 + 0.05\sin\theta$ and a polar radius of $0.2 - 0.1\sin\theta$. The prescribed motion induces a flow with a Reynolds number of about 10.

The shape of the spheroid is maintained by tethers. At time t^n each fiber point $\mathbf{X}_j^n$ is tethered to its respective location on the spheroid $\mathbf{T}_j^n$. The sphere itself is discretized by triangulating a regular octahedron, yielding $N_B = 578$, 2502, and 10406 fiber points when $N = 32$, 64, and 128, respectively. When solving for $\mathbf{X}^{n+1}$ in the semi-implicit, lagged operators discretization, we take our initial guess to be $\mathbf{X}^{n+1,0} = \mathbf{T}^{n+1}$. We use the same 0.0001 convergence tolerance for CG as we do in the immersed plate simulation.

As in the previous example, we compare (explicit) FE/BE simulations with the proposed, fast, semi-implicit approach. For the FE/BE method the stable timestep is determined by the empirical formula $\Delta t = 0.5 \cdot 10^{-1} h\sigma^{-1/2}$. Table 4 provides a list of stable, explicit timesteps for this problem. The required Δt for a stable FE/BE simulation is even smaller than that in the plate example. Again, such direct, FE/BE simulations are impractical and would require a massive computational effort even for modest resolutions. For the implicit, lagged operators discretization it is again sufficient to fix $\Delta t = 0.002$ to maintain both accuracy and stability for all resolutions, $N =$ 32, 64, and 128. A comparison of the total CPU time for all simulations is presented in Table 5. As in the plate example, the CPU time for the fast, semi-implicit simulations is almost invariant as σ increases. The numbers are even more striking than in the preceding example; for $N = 128$ and $\sigma = 10^9$, the proposed approach is six orders of magnitude faster than the commonly used FE/BE approach. A depiction of the flow obtained using the new fast,

Table 5: Total CPU time in hours for the explicit and semi-implicit simulations of the immersed spheroid, with varying values of σ. * denotes an extrapolated value.

		$\sigma = 10^5$	$\sigma = 10^6$	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$
$N =$ 32	Implicit	0.013	0.015	0.015	0.016	0.016
	Explicit	4.405	13.675	43.467	135.742*	429.701*
$N =$ 64	Implicit	0.127	0.134	0.140	0.138	0.137
	Explicit	83.155*	253.626*	822.769*	2595.119*	8213.271*
$N =$ 128	Implicit	1.057	1.125	1.141	1.151	1.167
	Explicit	1326.0*	4170.7*	13904.8*	43827.9*	134516.2*

semi-implicit approach is presented in Figure 8.

6.3. Accuracy

There are a number of differences between our implicit methodology and the standard FE/BE explicit methodology. Each difference is a potential source of additional numerical error: the larger timestep taken, the implicit discretization itself, the assumption of translation invariance of G_h, the trilinear interpolation used to calculate values of the form $G_h(\mathbf{z})$, the far field expansion of G_h, and approximate nature of the Krylov subspace solvers. We have been extra careful to ensure that each of these errors is no more than $O(h)$, the underlying order of the IB Method, and to verify numerically that these errors do not accumulate.

To analyze error accumulation we again turn to our simulations of a plate and a sphere. For the following simulations we fix the immersed structure via tether points with spring constant $\sigma = 10^6$ and induce a simple flow by adding a constant force $\mathbf{f} = (0, 0, 1)$ to every point of the fluid domain. We perform the simulation first for $N = 128$ using the explicit method and store the resulting fluid velocity at time $T = 0.1$ as u_{128}. This velocity field will serve as a our standard by which we gauge the accuracy of other simulations.

We now perform the same simulation twice for $N = 32$, 64, and 96, once using the explicit method and once using our implicit methodology. We calculate the relative error between a velocity field $\tilde{u}$ and our standard u_{128} by taking the l^2-norm, $\|u_{128} - \tilde{u}\|_{l^2} / \|u_{128}\|_{l^2}$. For simplicity we downsample all velocity fields to a $32 \times 32 \times 32$ grid. The resulting errors can be seen in Table 6.

The relative errors from our implicit simulations are roughly the same as those from the explicit simulations, confirming that our fast implicit methodology is not generating any unacceptable inaccuracies.

Finally, we perform one more test to verify that approximating G_h as translation invariant does not lead to a degradation of the overall accuracy. We do a simulation using an identical setup as that used for the calculation of u_{128}, except we shift the immersed boundary upward by a distance of $h/2$. We store the resulting fluid velocity at time $T = 0.1$ as $\tilde{u}_{128}$.

If G_h were exactly translation invariant then the resulting simulation would be identical to the original simulation up to a shift of the velocity field. That is, we would have $u_{128}(\mathbf{z}) = \tilde{u}_{128}(\mathbf{z} + (0, 0, h/2))$, for any $\mathbf{z} \in \Omega_h$. This is the case with the continuous equations.

For our discrete simulations G_h is not exactly translation invariant and there is a difference between $u_{128}(\mathbf{z})$ and $\tilde{u}_{128}(\mathbf{z} + (0, 0, h/2))$. Note that we use simple linear interpolation to calculate values of $\tilde{u}_{128}$ between grid points. Calculating the relative difference between u_{128} and a shifted $\tilde{u}_{128}$ using the l^2-norm as before, we see that this difference is less than 1%, or smaller than the error of the method. Shifts other than $(0, 0, h/2)$ yield similar results.

7. Conclusion

In [**?**], novel expedited methods for the semi-implicit system (7)-(9) in 2D were proposed. The cost of the implicit solvers therein presented were on the order of the cost of a fluid solve, allowing for efficient implicit timesteping with computational cost on the same order as that of an explicit timestep. The direct extension of that methodology to 3D proved unfeasible due to the large number of fiber points common to 3D IB applications.

In this paper, we presented an entirely new, alternative methodology suitable for the 3D case and for when there is a large number of immersed boundary point. We showed that the efficiency of the proposed fast semi-implicit solver, relative to the cost of a fluid solve, is asymptotically superior to the solvers in the 2D case. Indeed, in our test problems, we demonstrated

Table 6: The relative error between various low resolution simulations and a high resolution explicit simulation.

		$N = 32$	$N = 64$	$N = 96$
Plate	Implicit	0.160	0.0327	0.015
	Explicit	0.222	0.073	0.024

		$N = 32$	$N = 64$	$N = 96$
Sphere	Implicit	0.058	0.015	0.005
	Explicit	0.089	0.029	0.009

that solving the implicit system was not the predominant cost for the semi-implicit timestep, but was rather overshadowed by the cost of the necessary fluid solves. Thus, we have shown that the stiffness inherent in many IB applications can be eliminated via a robust, semi-implicit discretization for a minimal cost. More importantly, the proposed approach scales very well as N_B increases, allowing the new methodology to be applied to a wide range of complex structures. The computational savings obtained with the new methodology are enormous. IB Method computations that would typically require weeks or months using the standard, FE/BE approach can now be performed in just minutes.

Acknowledgments

The authors would like to thank Profs. Alexandre Roma and Paul J. Atzberger for insightful conversations about this work. Partial support for this work was provided by the National Science Foundation under Grant number DMS 1016310.

Appendix A. The octree

Note that for this appendix we adopt a more programmatic syntax, replacing Ω_{in}^{P} with $P.\Omega_{in}$, $\mathbf{c}^P$ with $P.\mathbf{c}$, and so on. This is to facillitate the exposition of pseudocode.

We start by defining the notion of a *panel*. A panel is simply a cubic subset of Ω, together with an optional collection of 8 child panels that divide the parent into equal octants. Let P be a panel. To P we associate three vectors, $P.\mathbf{c}, P.\mathbf{a}, P.\mathbf{b} \in \Omega$. $P.\mathbf{a}$ and $P.\mathbf{b}$ serve to define the domain of P, given by

$$P.\Omega_{out} = [P.\mathbf{a}_0, P.\mathbf{b}_0] \times [P.\mathbf{a}_1, P.\mathbf{b}_1] \times [P.\mathbf{a}_2, P.\mathbf{b}_2] \subseteq \Omega. \tag{A.1}$$

The center of P is denoted $P.\mathbf{c} = (P.\mathbf{a} + P.\mathbf{b})/2$. This will be the center of expansion for poles inside P. We also associate to P a collection of fiber points $P.\mathbf{Y}$ consisting of those points of the fiber configuration $\mathbf{X}$ lying within the domain $P.\Omega_{out}$. The children of P, if it has any, are denoted by $P.Child[i]$, for $0 \leq i < 8$. The children divide $P.\Omega_{out}$ equally, by splitting it into 8 pieces via the 3 planes $x = P.\mathbf{c}_0$, $y = P.\mathbf{c}_1$, and $z = P.\mathbf{c}_2$.

Given a domain $\Omega = [0,1]^3$, we define an initial panel called *Root*, specified by the bounds $Root.\mathbf{a} = (0,0,0)$ and $Root.\mathbf{b} = (1,1,1)$. We now recursively define child panels, creating the 8 children of *Root*, then the 8 children of each of those children, and so on. For a panel P with few fiber points in its domain, say $|P.\mathbf{Y}| < MinPoints$, we truncate the recursive process, leaving P childless. For a pseudocode version of this process, see Function Appendix A.1 CreatePanel. For our simulations we take $MinPoints = 10$. A rigorous accounting of the treecode costs can lead to an optimal value of $MinPoints$, but we did not pursue such an analysis here.

The totality of *Root* and all of its branches is collectively known as the octree. We now introduce the concept of *well separatedness*. Given a point $\mathbf{x}$ and a panel P we say that $\mathbf{x}$ is well separated from P provided that

$$|\mathbf{x}_i - P.\mathbf{c}_i| \geq 3(P.\mathbf{c}_i - P.\mathbf{a}_i), \qquad \text{for } i = 0,1,2, \tag{A.2}$$

that is, provided that $\mathbf{x}$ is sufficiently far from the center of P. We now consider the pair of subsets $(P.\Omega_{in}, P.\Omega_{out})$, where $P.\Omega_{in}$ is the collection of points in Ω that are well separated from P. We assume that we can expand G_h over $P.\Omega_{in} \times P.\Omega_{out}$, arriving at two collections of coefficient functions $\{P.A_k\}_{k=1}^{p}$ and $\{P.B_k\}_{k=1}^{p}$. In addition, we define

$$P.B_{out} = \{i \in B | \mathbf{X}_i \in P.\mathbf{Y}\}, \tag{A.3}$$

as the collection of indices associated with fiber points in $P.\mathbf{Y}$.

Suppose now that we are given a fiber force $\mathbf{F}$ and wish to evaluate its influence at points $\mathbf{x}$ in the fluid domain. For each $\mathbf{x}$ this influence is given by

$$\sum_{1\leq j\leq N_B} G_h(\mathbf{x}-\mathbf{X}_j)\mathbf{F}_j. \tag{A.4}$$

To evaluate this efficiently, we first loop over each panel P and calculate the far field expansion of all the poles located in $P.\Omega_{out}$. The k-th term of this expansion is given by

$$P.H_k \equiv \sum_{j\in P.B_{out}} P.B_k(\mathbf{X}_j)\tilde{F}_j. \tag{A.5}$$

If $\mathbf{x}$ is well separated from P, then the incoming effect on $\mathbf{x}$ from P is given by

$$\mathbf{E}^T\sum_{k=1}^{p} P.A_k(\mathbf{X}_i)P.H_k. \tag{A.6}$$

We can now evaluate the entire influence of $\mathbf{F}$ on a point $\mathbf{x}$. The process is best described in recursive form. A pseudocode is presented in Function Appendix A.2 Evaluate.

Because our treecode must be invoked multiple times per timestep, it is critical to streamline its evaluation. There are two key points to keep in mind. First, calculations of the expansion coefficients can be recycled. In Section 5.3 we will detail the form of the coefficient functions A_k and B_k. Each evaluation requires an expensive trilinear interpolation. Avoiding duplications of these calculations provides substantial speedup. Second, using the treecode to evaluate the influence at a particular point $\mathbf{x}$ requires traversing the octree. This traversal can be stored as a template so that additional evaluations at $\mathbf{x}$ do not require a recursive call.

There are thus two different function calls: the actual evaluation function, which takes in a fiber force $\mathbf{F}$ and returns the influence over the entire structure $\mathbf{X}$; and a pre-evaluation function, which does a variety of computations used to streamline the evaluation function. The cost of the pre-evaluation function is much higher than that of the evaluation function itself. However, the pre-evaluation function is called only once per timestep.

We mention one further implementation detail regarding the treatment of the domain $P.\Omega_{in}$ for a given panel P. Rather than treating this is as a single domain, we break it into eight equally sized pieces, corresponding to the eight octants around the panel's center $P.\mathbf{c}$. For each piece we must then have separate collections of coefficient functions $\{P.A_k\}_{k=1}^p$, $\{P.B_k\}_{k=1}^p$, as well as a separate expansion $\{P.H_k\}_{k=1}^p$.

This additional detail may seem costly at first, but in fact it increases the speed of the treecode substantially. By restricting our attention to a smaller domain it is possible to find expansions which converge much more rapidly.

function Appendix A.1 CreatePanel($\mathbf{a}, \mathbf{b}$)

```
  $P \leftarrow$ new $Panel$
  $P.\mathbf{a} = \mathbf{a}$
  $P.\mathbf{b} = \mathbf{b}$
  $P.\mathbf{c} = (\mathbf{a}+\mathbf{b})/2$
  if $P.Parent$ exists then
      $P.\mathbf{Y} = P.Parent.\mathbf{Y} \cap P.\Omega_{out}$
  else
      $P.\mathbf{Y} = \mathbf{X}^n \cap P.\Omega_{out}$
  if $|P.\mathbf{Y}| > MinPoints$ then
      $P.Child[0] =$ CreatePanel($P.\mathbf{a}, P.\mathbf{c}$)
      $P.Child[1] =$ CreatePanel($[P.\mathbf{c}_0, P.\mathbf{a}_1, P.\mathbf{a}_2], [P.\mathbf{b}_0, P.\mathbf{c}_1, P.\mathbf{c}_2]$)
      $P.Child[2] =$ CreatePanel($[P.\mathbf{a}_0, P.\mathbf{c}_1, P.\mathbf{a}_2], [P.\mathbf{c}_0, P.\mathbf{b}_1, P.\mathbf{c}_2]$)
      $P.Child[3] =$ CreatePanel($[P.\mathbf{c}_0, P.\mathbf{c}_1, P.\mathbf{a}_2], [P.\mathbf{b}_0, P.\mathbf{b}_1, P.\mathbf{c}_2]$)
      $P.Child[4] =$ CreatePanel($[P.\mathbf{a}_0, P.\mathbf{a}_1, P.\mathbf{c}_2], [P.\mathbf{c}_0, P.\mathbf{c}_1, P.\mathbf{b}_2]$)
      $P.Child[5] =$ CreatePanel($[P.\mathbf{c}_0, P.\mathbf{a}_1, P.\mathbf{c}_2], [P.\mathbf{b}_0, P.\mathbf{c}_1, P.\mathbf{b}_2]$)
      $P.Child[6] =$ CreatePanel($[P.\mathbf{a}_0, P.\mathbf{c}_1, P.\mathbf{c}_2], [P.\mathbf{c}_0, P.\mathbf{b}_1, P.\mathbf{b}_2]$)
      $P.Child[7] =$ CreatePanel($P.\mathbf{c}, P.\mathbf{b}$)
  return $P$
```

Appendix B. Computing the far field expansions

Appendix B.1. Precomputing the G_h lookup table

In Section 5, we derived the componentwise definition of $\mathcal{M}_n$ as

$$(\mathcal{M}_n\mathbf{F})_i \approx \sum_{0\le j\le N_B} G_h(\mathbf{X}_j - \mathbf{X}_i)\mathbf{F}_j, \qquad \text{for } 0 \le i \le N_B. \qquad \text{(B.1)}$$

This requires evaluation of the function G_h, defined in (28). To perform this evaluation efficiently we first precompute the values of G_h on our Eulerian

function Appendix A.2 Evaluate($\mathbf{x}, P$)

```
if $\mathbf{x} \in P.\Omega_{in}$ then {$\mathbf{x}$ is well separated from $P$}
    return $\mathbf{E}^T \sum_{k=1}^{p} P.A_k(\mathbf{X}_i) P.H_k$
else
    if $P$ has children then
        $S = 0$ {the $3 \times 3$ zero tensor}
        for $i = 0$ to 7 do
            $S = S +$ Evaluate($\mathbf{x}, P.Child[i]$)
        return S
    else {do a direct summation}
        return $\sum_{j \in P.B_{out}} G_h(\mathbf{x}, \mathbf{X}_j)\mathbf{F}_j$
```

grid $\mathcal{G}_h$. We refer to the resulting tensor field over $\mathcal{G}_h$ as a lookup table. The steps for creating the lookup table are as follows.

1. First, we spread a unit force $\mathbf{e}_i$ at the origin, obtaining a force field defined as $\mathbf{e}_i\delta(\mathbf{x})$ at any $\mathbf{x} \in \mathcal{G}_h$.
2. We calculate the influence of this force field on the fluid velocity by applying the operator $\mathcal{L}_h$, obtaining the vector field $\mathbf{u}_\mathbf{0}^i \equiv \mathcal{L}_h(\mathbf{e}_i\delta_h)$.
3. We spread the velocity $\mathbf{u}_\mathbf{0}^i$ at every point $\mathbf{x} \in \mathcal{G}_h$, defining

$$(G_h(\mathbf{x}))_i \equiv \alpha h_B \sum_{\mathbf{z} \in \mathcal{G}_\Omega} \mathbf{u}_\mathbf{0}^i(\mathbf{z})\delta_h(\mathbf{z} - \mathbf{x})\, h^3. \tag{B.2}$$

4. Repeating the above for $i = 0, 1, 2$, we arrive at our 3×3 tensor lookup table G_h over $\mathcal{G}_h$.

We can now rapidly evaluate terms of the form $G_h(\mathbf{X}_j - \mathbf{X}_i)$ by looking up the value of $\mathbf{X}_j - \mathbf{X}_i$ on our lookup table. For vectors that do not lie exactly on an Eulerian intersection we make use of trilinear interpolation.

Appendix B.2. Calculating a truncated SVD of restrictions of G_h

The far field expansions of G_h that we employ are known as rank-1 decompositions, and can be obtained via the Singular Value Decomposition. Note that when we decompose G_h we consider it properly as a function defined on $\Omega \times \Omega$, not as a field over Ω. For actual evaluations of G_h, however, we still assume translation invariance, treating G_h as a field and making use of our lookup table on $\mathcal{G}_h$.

Treating G_h as a function defined only on $\mathcal{G}_h \times \mathcal{G}_h$, we may write G_h as an $N^3 \times N^3$ matrix and seek its SVD. Typical SVD algorithms would require $O(N^9)$ operations. This is too great a cost, even for a precomputation. The two key ingredients for accelerating the computation of the SVD are, first, that we only require the first p singular values of G_h, not all N^3 of them, and, second, that our approximation to G_h is convolutional. We present an algorithm that takes advantage of these ingredients. While matrices are convenient for relating to the SVD, in what follows we treat G_h directly as a function, and abandon any use of matrices.

We will focus on the discretized domain $\mathcal{G}_h$ and a particular pair of subsets, $(\Omega_{in}, \Omega_{out})$, over which we wish to decompose G_h. We will require a few definitions. Given two functions f and g defined over $\Omega_{in} \cap \mathcal{G}_h$ and $\Omega_{out} \cap \mathcal{G}_h$ respectively, we define the outer product $f \otimes g$ via the relation

$$(f \otimes g)(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})g(\mathbf{y}), \tag{B.3}$$

for $\mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h$, $\mathbf{y} \in \Omega_{out} \cap \mathcal{G}_h$. If f' is another function defined over $\Omega_{in} \cap \mathcal{G}_h$, and both f and f' are 3×3 tensor fields, then we define the componentwise inner product $\langle f, f' \rangle$ to be a 3×3 tensor given by

$$\langle f, f' \rangle_{ij} = \sum f_{ij}(\mathbf{x}) f'_{ij}(\mathbf{x}), \tag{B.4}$$

where the sum is taken over all $\mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h$.

For a scalar function f defined on $\mathcal{G}_h$, the L^2 norm is given by

$$\|f\|_2^2 = \sum_{\mathbf{x} \in \mathcal{G}_h} f(\mathbf{x})^2. \tag{B.5}$$

For functions with other domains the norm is implicitly taken over the entire domain of the function. For a 3×3 tensor valued function g on $\mathcal{G}_h$, we define $\|g\|_2$ to be a 3×3 tensor given by $(\|g\|_2)_{ij} = \|g_{ij}\|_2$.

Now, we are seeking a collection of $2p$ functions, $\{A_k\}_{k=1}^p$ and $\{B_k\}_{k=1}^p$, each a 3×3 tensor field over $\mathcal{G}_h$, such that the rank-1 decomposition

$$\sum_{k=1}^{p} A_k \otimes B_k \tag{B.6}$$

is as close as possible to G_h in the L^2 sense. From SVD theory we know that given this truncated decomposition we will have simultaneously minimized

$$\left\| G_h - \sum_{k=1}^{q} A_k \otimes B_k \right\|_2 \tag{B.7}$$

for any q such that $1 \leq q \leq p$, where the norm is taken over the domain $\Omega_{in} \cap \mathcal{G}_h \times \Omega_{out} \cap \mathcal{G}_h$. We may take advantage of this result by first finding A_1 and B_1 such that (B.7) is minimized for $q = 1$. Next, we can hold A_1 and B_1 fixed as we seek the A_2 and B_2 that minimize (B.7) for $q = 2$. Assuming we have iterated this procedure up to $q = l - 1$, we detail the procedure for finding A_l and B_l.

First, A_l and B_l are exactly the tensor fields that minimize

$$L \equiv \left\| G_h - \sum_{k=1}^{l} A_k \otimes B_k \right\|_2^2 = \left\| G'_h - A_l \otimes B_l \right\|_2^2 , \tag{B.8}$$

where

$$G'_h \equiv G_h - \sum_{k=1}^{l-1} A_k \otimes B_k. \tag{B.9}$$

If we fix B_l then we can minimize L in (B.8) by the method of least squares. We fix a $\mathbf{z}$ in Ω_{in} and derive L with respect to $A_l(\mathbf{z})$, obtaining

$$-\frac{1}{2}\frac{\partial L}{\partial A_l(\mathbf{z})} = -\frac{1}{2}\frac{\partial}{\partial A_l(\mathbf{z})} \sum \left(G'_h(\mathbf{x},\mathbf{y}) - A_l(\mathbf{x}) B_l(\mathbf{y})\right)^2 , \tag{B.10}$$

where the sum is taken over $\mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h, \mathbf{y} \in \Omega_{out} \cap \mathcal{G}_h$. Dropping the terms independent of $A_l(\mathbf{z})$ leaves

$$\begin{aligned} &\sum \left(G'_h(\mathbf{z},\mathbf{y}) - A_l(\mathbf{z}) B_l(\mathbf{y})\right) B_l(\mathbf{y}) \\ &= \left[\sum G'_h(\mathbf{z},\mathbf{y}) B_l(\mathbf{y})\right] - A_l(\mathbf{z}) \left\| B_l \right\|_2^2 , \end{aligned} \tag{B.11}$$

where the sum is now over $\mathbf{y} \in \Omega_{out} \cap \mathcal{G}_h$. The sum in (B.11) is nearly a convolution. We seek to recast it as such.

First, we define the indicator functions

$$\mathbf{I}_{in}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h \\ 0 & \text{otherwise,} \end{cases} \tag{B.12}$$

and

$$\mathbf{I}_{out}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \Omega_{out} \cap \mathcal{G}_h \\ 0 & \text{otherwise.} \end{cases} \tag{B.13}$$

A_l is a field over $\Omega_{in} \cap \mathcal{G}_h$, but we may identify $A_l\mathbf{I}_{in}$ with a field defined over all of $\mathcal{G}_h$ in the natural way. We may likewise identify $B_l\mathbf{I}_{out}$ with a field defined over all of $\mathcal{G}_h$. These identifications allow us to express the sum in (B.11) as a convolution against G_h over all of $\mathcal{G}_h$, minus some correction.

$$\begin{aligned} &\left[\sum G'_h(\mathbf{z},\mathbf{y})B_l(\mathbf{y})\right] \\ &= \left[\sum G_h(\mathbf{z}-\mathbf{y})B_l(\mathbf{y})\right] - \sum_{k=1}^{l-1}\sum A_k(\mathbf{z})B_k(\mathbf{y})B_l(\mathbf{y}) \\ &= \left[(G_h * (B_l\mathbf{I}_{out}))\mathbf{I}_{in}\right](\mathbf{z}) - \sum_{k=1}^{l-1} A_k(\mathbf{z})\left\langle B_k, B_l\right\rangle . \end{aligned} \tag{B.14}$$

Requiring that the derivatives of L be zero in (B.11), we solve for A_l as

$$A_l = \frac{1}{\|B_l\|_2^2}\left[(G_h * (B_l\mathbf{I}_{out}))\mathbf{I}_{in} - \sum_{k=1}^{l-1} A_k\left\langle B_k, B_l\right\rangle\right]. \tag{B.15}$$

If we fix A_l we can solve the equivalent least squares problem for B_l, obtaining

$$B_l = \frac{1}{\|A_l\|_2^2}\left[(G_h * (A_l\mathbf{I}_{in}))\mathbf{I}_{out} - \sum_{k=1}^{l-1} B_k\left\langle A_k, A_l\right\rangle\right]. \tag{B.16}$$

Equations (B.15) and (B.16) must be solved simultaneously. We approximate the solution to this system via the iterative method

$$\begin{cases} A_l^{n+1} = \dfrac{1}{\|B_l^n\|_2^2}\left[(G_h * (B_l^n\mathbf{I}_{out}))\mathbf{I}_{in} - \displaystyle\sum_{k=1}^{l-1} A_k\left\langle B_k, B_l^n\right\rangle\right], \\ B_l^{n+1} = \dfrac{1}{\|A_l^{n+1}\|_2^2}\left[(G_h * (A_l^{n+1}\mathbf{I}_{in}))\mathbf{I}_{out} - \displaystyle\sum_{k=1}^{l-1} B_k\left\langle A_k, A_l^n\right\rangle\right]. \end{cases} \tag{B.17}$$

We require an initial guess for this iteration, call it B_l^0, with the restriction that B_l^0 not be a linear combination of $\{B_k\}_1^{l-1}$.

(B.17) is a type of power method. Power methods are known to converge quickly provided the singular values of G_h are sufficiently separated. For our particular problem (B.17) converges in 20 to 30 iterations. A stopping criteria can be improvised based on $\left\| B_l^{n+1} - B_l^n \right\|$, but for simplicity we simply fix the number of iterations at 30.

We have written (B.17) in a form that allows for a minimum of computational effort. The convolutions against G_h may be efficiently computed in discrete Fourier space at an $O(N^3 \log N)$ cost, while the inner products require only $O(N^3)$ operations each. Thus, the total cost per iteration of (B.17) is $O(N^3 \log N)$. We require $O(\log N)$ decompositions, hence the total precomputational cost is $O(30pN^3 \log^2 N)$.

Appendix C. Translation Error

The efficiency of the proposed methodology relies partly on the approximate translation invariance of the discrete Green's function G_h, constructed in Section 5. In [**?**], it was shown that in 2D the error $|(G_h(\mathbf{x}, \mathbf{y}) - G_h(\mathbf{x} - \mathbf{y}))_{ij}|$ is smaller than $O(h)$. We sketch here the extension of this result to the 3D case. Note that there are some notational differences between this paper and [**?**].

First, we require a Green's function g_h associated with the operator $\mathcal{L}_h$. G_h, the Green's function relating the influence of one fiber point on another, can be written as a summation involving g_h. Bounds on g_h will lead to bounds on the translation error associated with G_h.

We define three force fields

$$\mathbf{f}_i(\mathbf{x})_j = \begin{cases} 1/h^3 & \text{if } \mathbf{x} = \mathbf{0}, i = j \\ 0 & \text{otherwise,} \end{cases} \tag{C.1}$$

for $i, j = 1, 2, 3$. These force fields correspond to a suitably scaled point force along a cardinal direction centered at the origin. With these we define the 3×3 tensor field

$$g_h(\mathbf{x})_{ij} = \mathcal{L}_h(\mathbf{f}_i)_j. \tag{C.2}$$

Following Lemma 5.1 in [**?**] we may bound the components of g_h via

$$\begin{aligned} \|(g_h)_{ij}\|_\infty &\leq \sum \frac{1}{1 + 16\nu\Delta t|\mathbf{k}|^2} \\ &\leq C \int_0^{\frac{\sqrt{2}}{2}N} \frac{1}{1 + 16\nu\Delta t r^2} r^2 \, dr < C\frac{N}{\Delta t}, \end{aligned} \tag{C.3}$$

where the sum is over the set $\{\mathbf{k} \in \mathbb{Z}^3 \mid |\mathbf{k}_i| < N/2\}$, and the integral is obtained from a spherical coordinate transformation.

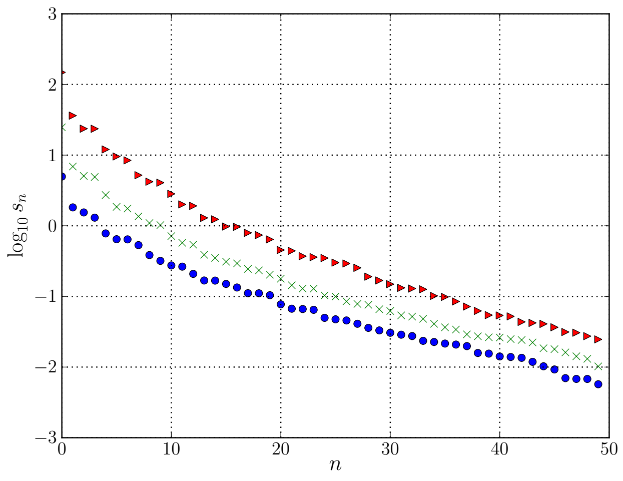
Consider now two fiber points, located at $\mathbf{x}$ and $\mathbf{y}$. We wish to bound the tensor components of the error $G_h(\mathbf{x}, \mathbf{y}) - G_h(\mathbf{x} - \mathbf{y})$. The bound given in Theorem 5.1 in [**?**] tells us that

$$|(G_h(\mathbf{x}, \mathbf{y}) - G_h(\mathbf{x} - \mathbf{y}))_{ij}| < \frac{2(\Delta t)^2}{\rho} \left\|(g_h)_{ij}\right\|_\infty h_B. \tag{C.4}$$

Assuming $\Delta t \propto h$ and $h_B \propto h^2$, then (C.4) combined with (C.3) gives us the bound

$$|G_h(\mathbf{x}, \mathbf{y}) - G_h(\mathbf{x} - \mathbf{y})|_{ij} < Ch^2, \tag{C.5}$$

which is smaller than the $O(h)$ error of the IB Method.



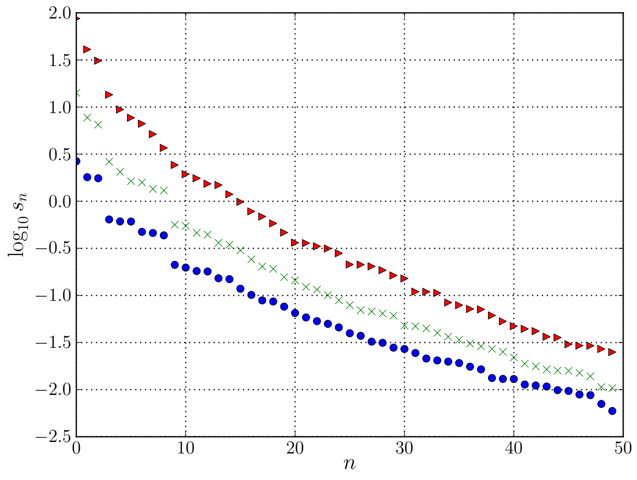


Figure 2: Singular values of the discrete Green's function: $(G_h)_{xx}$ component (top) and $(G_h)_{xy}$ component (bottom). 'o' markers are for $N = 32$, 'x' markers are for $N = 64$, and '$\triangle$' makers are for $N = 128$.

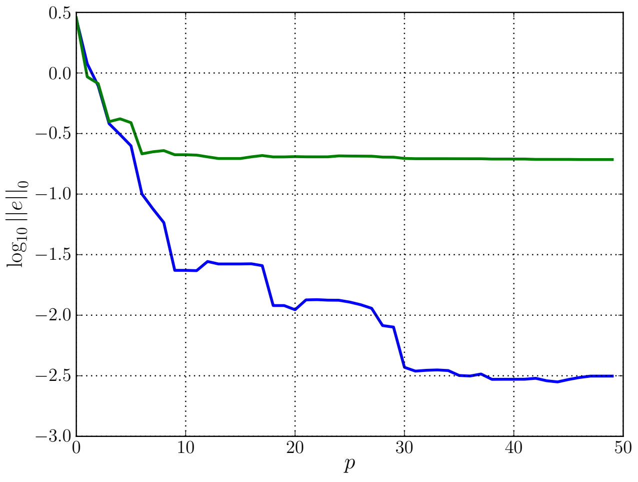


Figure 3: Log of the sup norm of the difference between $\mathcal{M}_n\mathbf{F}$ calculated via a fluid solve and via the treecode for various values of p, where p is the number of terms in the treecode's far field expansion. $N = 128$ and $N_B = N^2/4$. The blue line is for a plate with fiber points aligned to the Eulerian grid. The green line is for a plate with no restrictions on the fiber point locations.

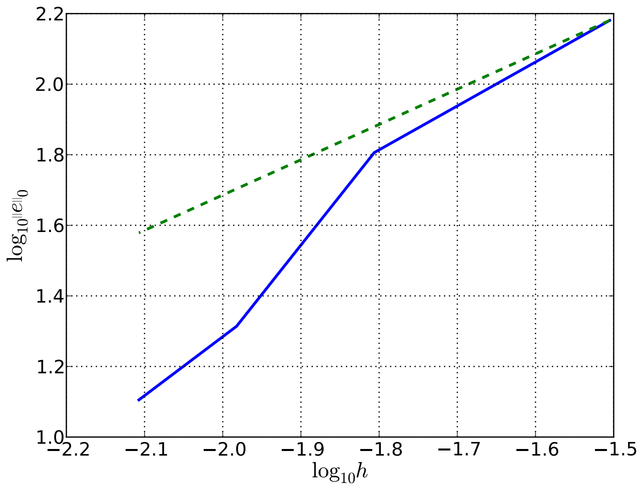


Figure 4: Log of the sup norm of the difference between $\mathcal{M}_n\mathbf{F}$ calculated via a fluid solve and via the treecode for various values of h, with $p = 10$. The solid blue line is the error. The dashed green line is a reference line with slope 1.

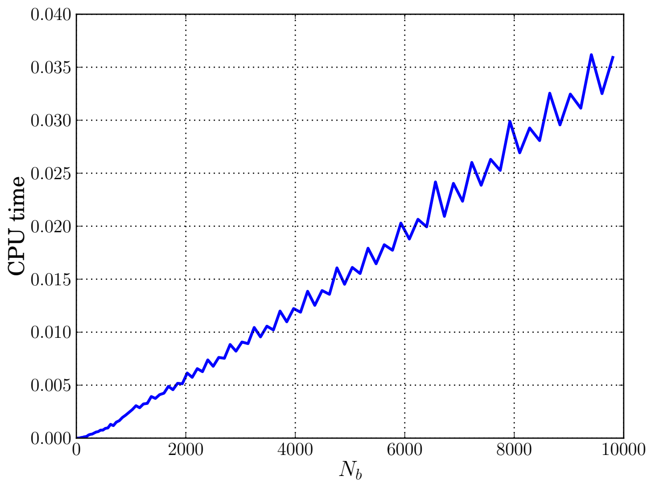


Figure 5: CPU time for the evaluation of $\mathcal{M}_n\mathbf{F}$ via the treecode for increasing values of N_B, where $p = 10$. Time is given as multiples of the average time to perform a fluid solve, for $N = 128$.

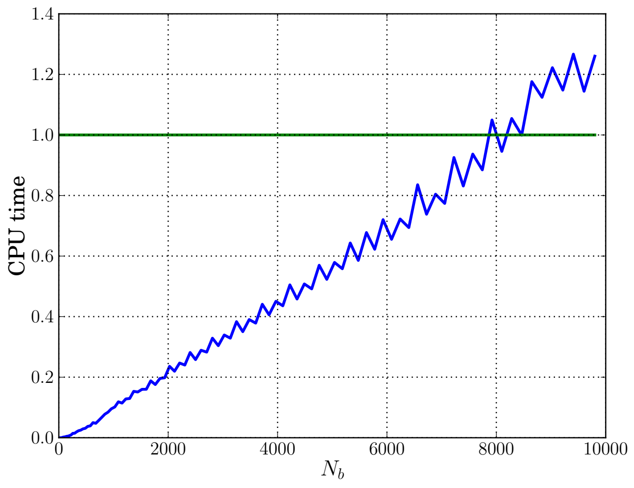


Figure 6: CPU time for the treecode pre-evaluation for increasing values of N_B, where $p = 10$. Time is given as multiples of the average time to perform a fluid solve, for $N = 128$. The green line represents the cost of one fluid solve.

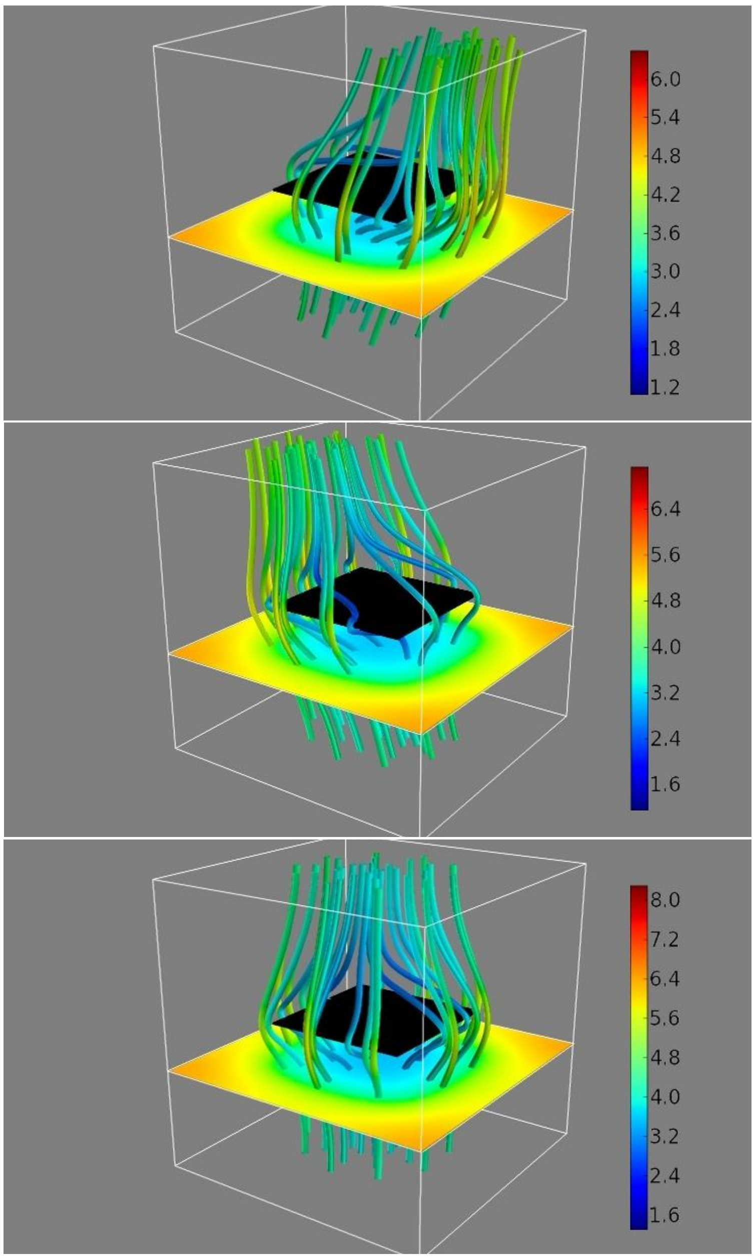


Figure 7: Plot of the immersed plate with flow lines. The plate is drawn in black. Below the plate is a cross-section of the z-component of the velocity field with an associated color bar. $N = 128$, and $N_B = 8192$. Frames shown at total simulation time $T = 0.020$, 0.142, and 0.208, from top to bottom.

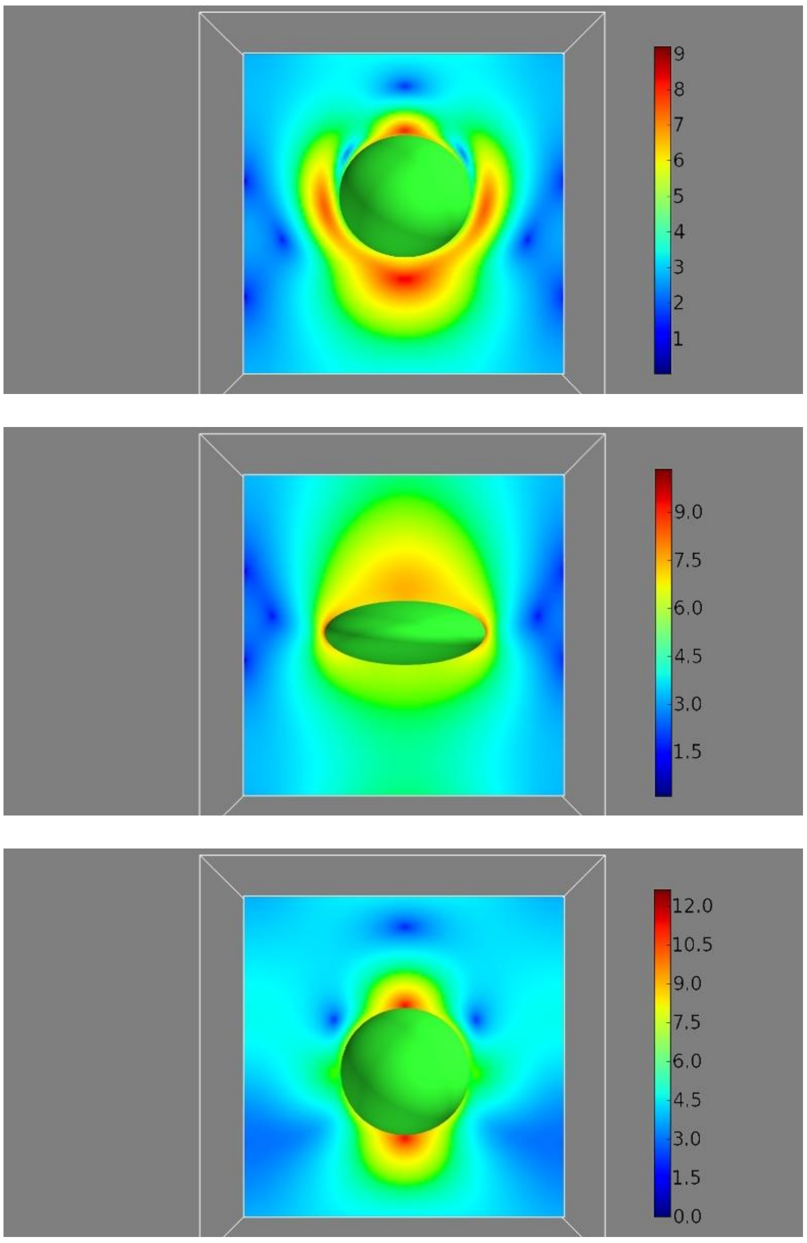


Figure 8: Plot of the immersed spheroid, drawn in green. Cutting the spheroid is a cross-section of the velocity magnitude scalar field with an associated color bar. $N = 128$, and $N_B = 10406$. Frames shown at total simulation time $T = 0.006$, 0.062, and 0.126, from top to bottom.