Efficient tools for the design and simulation of microelectromechanical and microfluidic systems

by

Carlos Pinto Coelho

Eng., Instituto Superior Técnico, Universidade Técnica de Lisboa (1999)
M.Eng., Instituto Superior Técnico, Universidade Técnica de Lisboa (2001)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2007

Author . . . .
Department of Electrical Engineering and Computer Science
August 10, 2007

Certified by . . . .
Jacob K. White
Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Efficient tools for the design and simulation of microelectromechanical and microfluidic systems

by

Carlos Pinto Coelho

Submitted to the Department of Electrical Engineering and Computer Science
on August 10, 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In air-packaged surface micromachined devices and microfluidic devices the surface to volume ratio is such that drag forces play a very important role in device behavior and performance. Especially for surface micromachined devices, the amount of drag is greatly influenced by the presence of the nearby substrate. In this thesis a precorrected FFT accelerated boundary element method specialized for calculating the drag force on structures above a substrate is presented. The method uses the Green's function for Stokes flow bounded by an infinite plane to implicitly represent the device substrate, requiring a number of modifications to the precorrected FFT algorithm. To calculate the velocity due to force distribution on a panel near a substrate an analytical panel integration algorithm was also developed. Computational results demonstrate that the use of the implicit representation of the substrate reduces computation time and memory while increasing the solution accuracy. The results also demonstrate that surprisingly, and unfortunately, even though representing the substrate implicitly has many benefits it does *not* completely decouple discretization fineness from distance to the substrate.

To simulate the time dependent behavior of micromechanical and microfluidic systems, a stable velocity implicit time stepping scheme coupling the precorrected FFT solver with rigid body dynamics was introduced and demonstrated. The ODE library was integrated with the solver to enable the simulation of systems with collisions, contacts and friction. Several techniques for speeding up the calculation of each time step were presented and tested. The time integration algorithm was successfully used to simulate the behavior of several real-world microfluidic devices.

Thesis Supervisor: Jacob K. White
Title: Professor

Acknowledgments

First I would like to thank my advisor Professor Jacob White for his support and guidance through the years. Jacob has been both an advisor and friend always keeping my best interests in mind and showing true concern about my wellbeing and that of my family.

I would also like to thank Professor Miguel Silveira who, even from Portugal, has always been present as a friend and as a source of advice and encouragement.

I would also like to thank all my friends at MIT and elsewhere with whom I shared good times and bad. You know who you are.

Finally, and most importantly, I must thank my wife and family for their love and support. Without them I would not have made it here today.

Contents

Contributions

- Adapted the precorrected FFT to support non-translational invariant substrate Stokes flow Greens function.
- Coupled velocity implicit time stepping scheme to accelerated boundary element solver that enables the stable and efficient simulation of time dependent problems in microfludics.
- Developed analytical panel integration algorithm for polynomial force distributions over odd powers of the distance between the source and the evaluation point. This algorithm can be used to calculate the Stokes velocity field due to polynomial force and force multipole distributions on flat panels.
- Used C++ template metaprogramming techniques to implement efficient and generic routines that enable exploiting kernel symmetry to reduce memory requirements for the precorrected FFT algorithm.
- Developed specializations of the precorrected FFT algorithm for the calculation of the drag force on surface micromachined devices.
- Demonstrated the surprising result that using the Stokes substrate Green's function does *not* decouple structure discretization from distance to the substrate, regardless of the smoothness of the force distribution.

Chapter 1

Introduction

For small length scales, as the ratio between the surface area and the volume increases, drag forces play an important role in the behavior of any objects moving in a fluid. For the length scales and velocities encountered in many micromechanical and microfluidic applications, the Stokes flow model is known to produce accurate estimates of the drag forces on objects in a fluid [1, 2]. The Stokes drag force on air-packaged microelectromechanical systems (MEMS) such as oscillators, accelerometers and micromirrors is an important factor that significantly influences their dynamic behavior and performance [3, 4, 5, 6]. Especially for surface micromachined devices, the drag is greatly influenced by the presence of the nearby substrate [7]. In microfluidic devices the fluid drag force drives the motion of beads and cells in the flow and is also very important.

Several methods exist for the calculation of the drag forces on objects immersed in Stokes flow: finite differences [8], immersed boundary methods [9], the finite element method [10] and the boundary element method [1]. Since, for Stokes flow, the fluid structure only depends on the boundary configuration at the time point of interest, the boundary element method is a particularly suitable approach. Moreover, for problems where one is interested in the time domain evolution of a system, the boundary element method has the advantage that remeshing the domain at each step is not necessary. Furthermore, using the boundary element method with appropriate Green's functions it is often possible to drive the motion of the objects in the flow by specifying a background flow without having to explicitly discretize the surface of the microfluidic channel or other boundaries that, in other methods, would just be used to drive the bulk fluid.

The formulation of Stokes flow problems as boundary integral equations can be found in [11, 12, 1] and is reviewed in Chapter 2. Boundary element methods, based on discretization of the boundary integral equations for Stokes flow are briefly reviewed in Chapter 2. An analytical panel integration scheme for calculating the Stokes velocity due to a force distribution on a flat panel is presented in Chapter 3. However, naïve implementations of the boundary element method have a prohibitively high cost in both computation time and memory when applied to large engineering problems. Accelerated boundary element solvers based on the multipole method [13], panel clustering and wavelets [14] and on the precorrected FFT method [15], have been applied to the calculation of the Stokes drag force [16, 17, 18, 6, 19, 20].

In this thesis we present a precorrected FFT accelerated boundary element method specialized for calculating the drag force on structures above a substrate. Our method uses the Green's function for Stokes flow bounded by an infinite plane to implicitly represent the device substrate, requiring a number of modifications to the precorrected FFT algorithm. Computational results demonstrate that the use of the implicit representation of the substrate reduces computation time and memory while increasing the solution accuracy. The modified precorrected FFT algorithm and results demonstrating its use are presented in Chapter 4.

Our computational results demonstrate that, surprisingly, even though representing the substrate implicitly has many benefits, it does *not* completely decouple discretization fineness from distance to the substrate. A detailed description of this important result can be found in Chapter 5.

The calculation of the trajectories of objects moving in Stokes flow is a convenient tool for the design of microfluidic devices such as cell traps [21, 22, 23, 24] and micromixers [25]. Time domain simulation is also very important for the design of MEMS devices such as micromirrors [26].

The Stokes equations state that the pressure, viscous forces and body forces are at balance regardless of the history of flow, even though the boundaries of the flow maybe changing in time [1]. When there are no abrupt changes in the fluid velocity, momentum diffuses throughout the fluid domain much faster than the configuration of the flow is changing due to the evolution of its boundaries [1]. Therefore, in these conditions, a quasi-static approach for analyzing the time evolution of the system is appropriate [25]. However, for small length scales, such as those present in MEMS and microfluidic devices, the ratio of the drag forces and the mass of the bodies is such that the time constant associated with transferring momentum between an object and the surrounding fluid is very small. For typical geometries, the time scale for momentum transfer between the objects and the fluid is much smaller than the timescale at which the objects move through the devices, which is usually the time scale of interest in simulation. The existence of the very small time scale for momentum diffusion makes the problem stiff and severely limits the step sizes that explicit time integration schemes can use.

To deal with stiffness without incurring the excessive cost of solving a non-linear equation for the forces on the surface of the object at each time step, we couple the boundary element Stokes solver with a time stepping scheme that updates the velocity implicitly and the position explicitly. Using this velocity implicit scheme allows for the stable simulation of the motion of objects using large time steps. To deal with problems involving collisions, contacts and friction we coupled our velocity-implicit time integration method with the freely available rigid body physics library ODE [27]. The quasistatic velocity-implicit time domain solver for Stokes flow is presented in Chapter 6 where it is applied to a set of real-world microfluidic problems.

For the implementation of the precorrected FFT solver, C++ template metaprogramming techniques [28, 29] were used to construct efficient and generic routines that enable exploiting the symmetry of the Stokes flow Green's function's to reduce memory usage. The use of C++ template metaprogramming techniques also enabled the generic implementation of most of the building blocks for the precorrected FFT algorithms in a way that makes it easy for a new solver, with a different kernel, to be developed. Details regarding the implementation of the more interesting blocks of the precorrected FFT algorithm are presented in

Chapter 7.

Thesis structure

The thesis is structured as follows: in Chapter 2, a review of the Stokes flow model and the formulation of Stokes flow problems as boundary integral equations is presented; in Chapter 3, an analytical panel integration scheme for calculating the Stokes velocity due to a force distribution on a flat panel is presented; in Chapter 4, the precorrected FFT method is reviewed and extended to support the Stokes substrate Green's function; in Chapter 5, a surprising result describing the dependency of the solution accuracy on the discretization of the structures and their distance to the substrate is presented; in Chapter 6, a velocity-implicit time stepping scheme for the stable and efficient simulation of the motion of objects in Stokes flow is presented; in Chapter 7, a set of relevant technical contributions are described. Finally, in Chapter 8, conclusions are drawn and future work is suggested. While Chapter 8 is a global conclusions chapter, some of the other chapters also have a local set of conclusions and suggestions for future work.

Chapter 2

Background

For many air-packaged surface micromachined devices and microfluidic devices, it has been verified that the characteristic velocity U, characteristic length L, density ρ and viscosity μ are such that the Reynolds number $\mathrm{Re} = UL\rho/\mu$ is small, the viscous term in the Navier-Stokes equations for moment conservation dominates over the inertial terms and the fluid motion can be accurately modeled by the combination of the Stokes equation

$$-\nabla P + \mu \nabla^2 \mathbf{u} = \nabla \cdot \boldsymbol{\sigma} = 0 \tag{2.1}$$

and the continuity equation

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

where $\mathbf{u}$ is the fluid velocity, P is the pressure, μ is the viscosity, and $\boldsymbol{\sigma}$ is the fluid stress tensor, which can be written elementwise as

$$\sigma_{ik} = -P\delta_{ik} + \mu \left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i} \right), \tag{2.3}$$

or in matrix form as

$$\boldsymbol{\sigma} = -P\mathbf{I} + \mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right), \tag{2.4}$$

where $\nabla \mathbf{u}$ is the Jacobian of the velocity.

The Stokes equations (2.1) can be obtained as an approximation of the Navier-Stokes equations for momentum conservation[1]

$$\rho\frac{\partial \mathbf{u}}{\partial t} + \rho\mathbf{u}\cdot\nabla\mathbf{u} = \mu\nabla^2\mathbf{u} - \nabla P \tag{2.5}$$

where ρ is the fluid density. The approximation can be justified by non-dimensionalizing (2.5)

$$\mathrm{Re}\left(\frac{1}{\mathrm{Sr}}\frac{\partial\tilde{\mathbf{u}}}{\partial\tilde{t}} + \tilde{\mathbf{u}}\cdot\tilde{\nabla}\tilde{\mathbf{u}}\right) = \tilde{\nabla}^2\tilde{\mathbf{u}} - \frac{\Pi L}{\mu U}\tilde{\nabla}\tilde{P}$$

where $\mathrm{Sr} = \tau U/L$ where τ is either an externally imposed time constant or, in its absence, is the convective time scale and $\mathrm{Sr} = 1$.

The Reynolds number is the ratio of the diffusive time constant for momentum in the fluid $\tau_D = \rho L^2/\mu$ and the convective time constant $\tau_C = L/U$ it states that the momentum diffuses through the fluid much faster than it is convected. However, if τ_C is small it also means that any objects moving in the fluid are doing so in such a way that the time constant associated with the changing boundary configuration is larger than the time constant associated with momentum diffusion. In other words, the fluid reaches a steady state momentum distribution much faster than the boundaries move; this justifies a quasi-static approach for time domain integration. Note that the quasi-static time evolution model is not valid if the fluid motion is starting or stopping or if there are any hard collisions, in which case τ maybe much smaller than the convective time scale.

2.1 Boundary Integral Equation Formulation

An integral equation formulation for the Stokes flow problem can be constructed using the Lorentz reciprocity identity [1]. The Lorentz reciprocity identity states that if $(\mathbf{u}_A, P_A)$ and $(\mathbf{u}_B, P_B)$ are the solutions of two Stokes flow problems, defined on the same geometric domain but with different boundary conditions, the corresponding velocities and stress

[1]Equations (2.1) and (2.5) are vector equations that expresses momentum conservation along each axis.

tensors are related by

$$\frac{\partial}{\partial x_j}(u_k^A \sigma_{kj}^B - u_k^B \sigma_{kj}^A) = \nabla \cdot (\mathbf{u}_A^T \boldsymbol{\sigma}_B - \mathbf{u}_B^T \boldsymbol{\sigma}_A) = 0, \tag{2.6}$$

wherever the solutions A and B are non-singular.

The Lorentz reciprocity identity can be used to construct a boundary integral equation for $\boldsymbol{\sigma}_B$ and $\mathbf{u}_B$ by choosing a problem A that has a known solution, integrating (2.6) over the volume of fluid domain V and using the divergence theorem to reduce the integral over the volume of the domain to an integral on its surface, ∂V,

$$\int_V \nabla \cdot (\mathbf{u}_A^T \boldsymbol{\sigma}_B - \mathbf{u}_B^T \boldsymbol{\sigma}_A) dV = \int_{\partial V} (\mathbf{u}_A^T \underbrace{\boldsymbol{\sigma}_B \mathbf{n}}_{\mathbf{f}_B} - \mathbf{u}_B^T \underbrace{\boldsymbol{\sigma}_A \mathbf{n}}_{\mathbf{f}_A}) dS = 0 \tag{2.7}$$

where $\mathbf{f}_{A/B}$ represent the force applied to the fluid at a point on the surface where the normal direction is $\mathbf{n}$, pointing away from the fluid.

A common choice for problem A is the free-space Stokes flow Green's function, i.e. the fluid velocity, stress and pressure field produced by applying a point force $\mathbf{g}$ at $\mathbf{x}_s$ to (2.1) and solving

$$\begin{cases} -\nabla P + \mu \nabla^2 \mathbf{u} = \delta(\mathbf{x} - \mathbf{x}_s)\mathbf{g} \\ \nabla \cdot \mathbf{u} = 0, \end{cases}$$

which yields

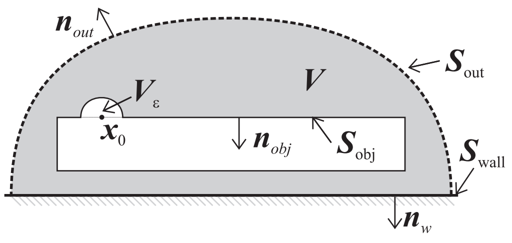
$$\mathbf{u}(\mathbf{x}) = \frac{1}{8\pi\mu}\mathbf{G}^F(\mathbf{x}, \mathbf{x}_s)\mathbf{g} = \frac{1}{8\pi\mu}\frac{1}{r}(\mathbf{I} + \hat{\mathbf{r}}\hat{\mathbf{r}}^T)\mathbf{g} \tag{2.8}$$

$$\boldsymbol{\sigma}(\mathbf{x}) = \frac{1}{8\pi}T_{ijk}^F(\mathbf{x}, \mathbf{x}_s)g_j = -\frac{3}{4\pi}\frac{1}{r^2}\hat{\mathbf{r}}\hat{\mathbf{r}}^T(\hat{\mathbf{r}}^T\mathbf{g}) \tag{2.9}$$

$$P(\mathbf{x}) = \frac{1}{8\pi}\mathbf{p}^F(\mathbf{x}, \mathbf{x}_s)\mathbf{g} = \frac{1}{4\pi}\frac{\hat{\mathbf{r}}}{r^2}\mathbf{g}$$

where $\mathbf{r} = \mathbf{x} - \mathbf{x}_s$, $r = \|\mathbf{r}\|_2$ and $\hat{\mathbf{r}} = \mathbf{r}/r$. The matrix relating the velocity field at $\mathbf{x}$ with the point force $\mathbf{g}$ at $\mathbf{x}_s$, $\mathbf{G}^F(\mathbf{x}, \mathbf{x}_s)$ in (2.8) is also known as a *stokeslet*. For simplicity, the ^{F} notation is dropped in this section.

Since the velocity $\mathbf{u}(\mathbf{x})$ in (2.8) and the stress tensor $\boldsymbol{\sigma}(\mathbf{x})$ in (2.9) are singular at $\mathbf{x}_s$, to apply (2.7) we exclude a region $V_\epsilon(\mathbf{x}_s)$ around $\mathbf{x}_s$ such that $\mathbf{u}^A(\mathbf{x})$ and $\boldsymbol{\sigma}^A(\mathbf{x})$ are analytical inside V as illustrated in Figure 2-1.



$\boldsymbol{n}_{out}$
$\boldsymbol{S}_{\text{out}}$
$\boldsymbol{V}$
$\boldsymbol{V}_\varepsilon$
$\boldsymbol{x}_0$
$\boldsymbol{n}_{obj}$
$\boldsymbol{S}_{\text{obj}}$
$\boldsymbol{S}_{\text{wall}}$
$\boldsymbol{n}_w$

Figure 2-1: *The integration volume V, in gray, is bounded by the substrate, the objects in the fluid, and an* infinite *surface S_{out}. The exclusion volume V_ϵ contains the source point $\mathbf{x}_s$ such that $\mathbf{u}_A(\mathbf{x})$ and $\boldsymbol{\sigma}_A(\mathbf{x})$ are analytical in $V - V_\epsilon$.*

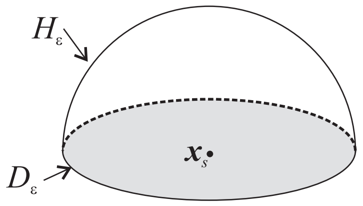


Figure 2-2: *When the source point $\mathbf{x}_s$ is located on a smooth surface, for small enough ϵ the exclusion region $V_\epsilon(\mathbf{x}_s)$ is bounded by a hemisphere H_ϵ and a disk D_ϵ.*

The exclusion volume $V_\epsilon(\mathbf{x}_s)$, which for convenience is often chosen to be a spherical section of radius $\epsilon > 0$, is parameterized on ϵ such that its surface and volume are asymptotically proportional to ϵ^2 and ϵ^3 respectively.

We obtain an equation for $\mathbf{u}_B$ and $\mathbf{f}_B$, which from now on we will refer to simply as $\mathbf{u}$ and $\mathbf{f}$, by applying (2.7) to $V - V_\epsilon$ which yields

$$\int_{\partial(V-V_\epsilon)} f_k(\mathbf{x})G_{ki}(\mathbf{x},\mathbf{x}_s) - \mu u_k(\mathbf{x})T_{kij}(\mathbf{x},\mathbf{x}_s)\mathbf{n}_j(\mathbf{x})dA = 0. \tag{2.10}$$

When the source point $\mathbf{x}_s$ is located on a smooth surface, for small enough ϵ the exclusion region $V_\epsilon(\mathbf{x}_s)$ is bounded by a hemisphere H_ϵ and a disk D_ϵ as illustrated in Figure 2-2. Using surfaces D_ϵ and H_ϵ, (2.10) can be rewritten as

$$\int\limits_{(\partial V - D_\epsilon)\bigcup H_\epsilon} f_k(\mathbf{x})G_{ki}(\mathbf{x},\mathbf{x}_s) - \mu u_k(\mathbf{x})T_{kij}(\mathbf{x},\mathbf{x}_s)\mathbf{n}_j(\mathbf{x})dA = 0 \tag{2.11}$$

That can be decomposed into a sum of simpler terms

$$\int_{\partial V} f_k(\mathbf{x}) G_{ki}(\mathbf{x},\mathbf{x}_s) dA = \mu \underbrace{\int_{\partial V} u_k(\mathbf{x}) T_{kij}(\mathbf{x},\mathbf{x}_s)\mathbf{n}_j(\mathbf{x}) dA}_{4\pi u_k(\mathbf{x}_s) \text{ for rigid body } \mathbf{u} \text{ and } \mathbf{x}_s \text{ on } \partial V} + \underbrace{\int_{D_\epsilon} f_k(\mathbf{x}) G_{ki}(\mathbf{x},\mathbf{x}_s)}_{O(\epsilon)\xrightarrow[\epsilon\to 0]{} 0}$$
$$- \underbrace{\int_{H_\epsilon} f_k(\mathbf{x}) G_{ki}(\mathbf{x},\mathbf{x}_s)}_{O(\epsilon)\xrightarrow[\epsilon\to 0]{} 0} - \mu \underbrace{\int_{D_\epsilon} u_k(\mathbf{x}) T_{kij}(\mathbf{x},\mathbf{x}_s)\mathbf{n}_j(\mathbf{x}) dA}_{=0 \text{ for } \mathbf{x}_s \text{ on disk}} + \mu \underbrace{\int_{H_\epsilon} u_k(\mathbf{x}) T_{kij}(\mathbf{x},\mathbf{x}_s)\mathbf{n}_j(\mathbf{x}) dA}_{\xrightarrow[\epsilon\to 0]{} 4\pi u_k(\mathbf{x}_s)} . \tag{2.12}$$

Considering only rigid body motion and then computing the limit of (2.12) as $\epsilon \to 0$ as the outer surface S_{out} stretches to infinity yields

$$\int_{\partial V} f_k(\mathbf{x}) G_{ki}(\mathbf{x},\mathbf{x}_s) dA = 8\pi\mu u_k(\mathbf{x}_s) \tag{2.13}$$

Using the symmetry relation $\mathbf{G}(\mathbf{x},\mathbf{x}_s) = \mathbf{G}^T(\mathbf{x}_s,\mathbf{x})$, true for any Stokes flow Green's function due to the Lorenz reciprocity theorem, and replacing the force on the fluid f_k by the force on the object surface, $-f_k$, results in

$$\int_{\partial V} G_{ik}(\mathbf{x}_s,\mathbf{x}) f_k(\mathbf{x}) dA = -8\pi\mu u_i(\mathbf{x}_s). \tag{2.14}$$

where the point $\mathbf{x}_s$, originally defined as the source of the Green's function $(\mathbf{u}_A, \boldsymbol{\sigma}_A)$, can now be interpreted as a *test point*.

2.2 Green's function for a flow bounded by a plane wall

The important forces in many MEMS are the forces acting on thin structures, S_{obj}, that are suspended over a substrate, S_{wall}, while the forces on the substrate do not usually determine device performance. The need to solve (2.14) explicitly for the force on the substrate can be eliminated by using a Green's function that satisfies a zero velocity condition on the substrate. Specifically, the Green's function can be computed by solving

$$\begin{cases} -\nabla P + \mu \nabla^2 \mathbf{u} = \delta(\mathbf{x} - \mathbf{x}_s)\mathbf{g} \\ \nabla \cdot \mathbf{u} = 0 \\ \mathbf{u}(\mathbf{x}) = \mathbf{0}, \quad \text{for } \mathbf{x} \text{ on } S_{\text{wall}}. \end{cases} \tag{2.15}$$

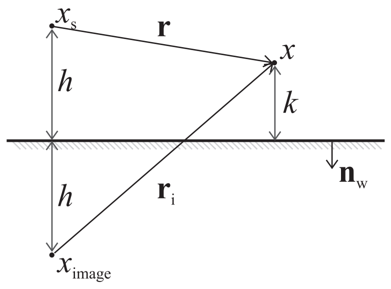


Figure 2-3: *Schematic representation of source and image source associated with the Green's function for Stokes flow bounded by a plane wall.*

In the presence of the substrate, S_{wall}, as illustrated in Figure 2-3, the solution of (2.15) is given by

$$\begin{aligned} \mathbf{u}(\mathbf{x}) = \mathbf{G}^{\text{w}}(\mathbf{x}, \mathbf{x}_s)\mathbf{g} = \mathbf{G}^{\text{F}}(\mathbf{x}, \mathbf{x}_s)\mathbf{g} - \mathbf{G}^{\text{F}}(\mathbf{x}, \mathbf{x}_i)\mathbf{g} \\ +2h^2\mathbf{G}^{\text{D}}(\mathbf{x}, \mathbf{x}_i)\mathbf{N}\mathbf{g} + 2h\mathbf{G}^{\text{SD}}(\mathbf{x}, \mathbf{x}_i)\mathbf{N}\mathbf{g} \end{aligned} \tag{2.16}$$

where h is the normal distance above the substrate of source point $\mathbf{x}_s$, $\mathbf{N} = \mathbf{I} - 2\mathbf{n}_{\text{w}}\mathbf{n}_{\text{w}}^T$, $\mathbf{n}_{\text{w}}$ is the wall unit normal as illustrated in Figure 2-3, $\mathbf{x}_i = \mathbf{N}\mathbf{x}_s$, $\mathbf{G}^{\text{F}}$ is the free-space Green's function (2.8) and

$$\mathbf{G}^{\text{D}}(\mathbf{x}, \mathbf{x}_i) = \frac{1}{r_i^3}(\mathbf{I} - 3\hat{\mathbf{r}}_i\hat{\mathbf{r}}_i)$$

is the potential dipole, where $\mathbf{r}_i = \mathbf{x} - \mathbf{x}_i$, $r_i = \|\mathbf{r}_i\|_2$, $\hat{\mathbf{r}}_i = \mathbf{r}_i/r_i$. The last term in (2.16) is referred to as the Stokeslet doublet and is given by

$$\mathbf{G}^{\text{SD}}(\mathbf{x}, \mathbf{x}_i) = \underbrace{(\mathbf{r}_i \cdot \mathbf{n}_{\text{w}})}_{-h-k}\mathbf{G}^{\text{D}}(\mathbf{x}, \mathbf{x}_i) + \underbrace{\frac{\hat{\mathbf{r}}_i\mathbf{n}_{\text{w}}^T - \mathbf{n}_{\text{w}}\hat{\mathbf{r}}_i^T}{r_i^2}}_{\mathbf{G}^{\text{R}}(\mathbf{r}_i)}. \tag{2.17}$$

Using the definition of $\mathbf{G}^{\text{SD}}$, $\mathbf{G}^{\text{D}}$ and $\mathbf{G}^{\text{R}}$, it follows that the substrate Green's function can be written as

$$\mathbf{G}^{\mathrm{S}}(\mathbf{x},\mathbf{x}_s) = \mathbf{G}^{\mathrm{F}}(\mathbf{r}) - \mathbf{G}^{\mathrm{F}}(\mathbf{r}_i) - 2hk\mathbf{G}^{\mathrm{D}}(\mathbf{r}_i)\mathbf{N} + 2h\mathbf{G}^{\mathrm{R}}(\mathbf{r}_i)\mathbf{N} \tag{2.18}$$

where k is the distance of the evaluation point to the substrate. If the plane normal $\mathbf{n}_{\mathrm{w}}$ is aligned with one of the global coordinate axes, $\mathbf{G}^{\mathrm{R}}$ has only two independent scalar entries, while $\mathbf{G}^{\mathrm{SD}}$ in (2.16) has 6 independent scalar entries. Reducing the number of unique scalar kernels can be used to reduce memory usage and computation time.

Using the substrate Green's function, (2.14) becomes

$$\int_{S_{\mathrm{obj}}} \mathbf{G}^{\mathrm{w}}(\mathbf{x}_s,\mathbf{x})\mathbf{f}(\mathbf{x})dA = -8\pi\mu\mathbf{u}(\mathbf{x}_s). \tag{2.19}$$

2.3 Nullspace and defect in the range

For any Green's function $\mathbf{G}(\mathbf{x}_s,\mathbf{x})$ associated with incompressible Stokes flow, the boundary integral equation for a single body (2.14) is singular and has a rank 1 nullspace given by $\mathbf{f} = \mathbf{n}$. The extension to the k-body case generates a rank-k nullspace [18]. This can shown by setting $\mathbf{f} = \mathbf{n}$ and using the divergence theorem

$$\int_{\partial V} G_{ik}(\mathbf{x}_s,\mathbf{x})n_k(\mathbf{x})dA = \int_{\partial V} G_{ki}(\mathbf{x},\mathbf{x}_s)n_k(\mathbf{x})dA = \int_V \frac{\partial G_{ki}(\mathbf{x},\mathbf{x}_s)}{\partial x_k}dA = \mathbf{0} \tag{2.20}$$

and recalling that $G_{ki}(\mathbf{x},\mathbf{x}_s) = \mathbf{G}(\mathbf{x},\mathbf{x}_s)\mathbf{e}_i$ is the k^{th} component of the velocity field due to a point force along the i^{th} direction applied on $\mathbf{x}_s$ and that $\nabla\mathbf{G}(\mathbf{x},\mathbf{x}_s)\mathbf{e}_i$ is the divergence of that incompressible velocity field, which is zero.

Due to the reciprocity relation, $\mathbf{G}(\mathbf{x},\mathbf{x}_s) = \mathbf{G}^T(\mathbf{x}_s,\mathbf{x})$, and therefore the defect in the range of the integral operator in (2.14) is also $\mathbf{u}(\mathbf{x}_s) = \mathbf{n}(\mathbf{x}_s)$. Therefore, for (2.14) to have a solution, the velocity field must satisfy

$$\int_{\partial V} \mathbf{n}^T(\mathbf{x})\mathbf{u}(\mathbf{x})dA = 0, \tag{2.21}$$

or equivalently, the net flux over the body surfaces S_{obj} must be zero. Fortunately, all the motion velocities $\mathbf{u}$ due to rigid body motion satisfy the zero net flux condition (2.21) and therefore (2.14) has a solution.

For the substrate case, since $\nabla \cdot (\mathbf{G}^{\text{w}}(\mathbf{x}, \mathbf{x}_s)\mathbf{g}) = 0$, for any force $\mathbf{g}$, and $\mathbf{G}^{\text{w}}(\mathbf{x}, \mathbf{x}_s) = \mathbf{G}^{\text{w,T}}(\mathbf{x}_s, \mathbf{x})$ the nullspace and defect of (2.19) are still the object surface normals $\mathbf{n}$.

There are several approaches to handling the nullspace problem [30, 6]. For the examples examined in this thesis we used the simplest approach in [6], computing the null-space free solution by projection.

2.4 Boundary Element Method

The Stokes flow problem defined by the integration volume V and a set of boundary conditions on $\mathbf{u}(\mathbf{x}_s)$ at each point $\mathbf{x}_s$ on ∂V is represented, in a continuous infinite dimensional form by (2.14) or (2.19). However, for almost any practical problem, there is no explicit analytical solution for (2.14) or (2.19). Approximate values for the forces on the object surface are generated by limiting the dimension of the solution space and the number of constraints to a finite number.

There are several ways to generate a finite linear system of equations from the boundary integral equation (2.14) or (2.19). In this section we describe one of the simplest possible discretization schemes: constant strength collocation. First the integration surface ∂V is discretized into a set of n_{panels} triangular or quadrilateral flat panels. The value of the velocity and the drag force on each panel is approximated by a constant value. With this discretization method, velocities and forces can be represented as vectors $\mathbf{U}$ and $\mathbf{F}$ in $\mathbb{R}^{3\times n_{\text{panels}}}$. To generate a set of $3n_{\text{panels}}$ equations using collocation, consider imposing

$$\sum_{j=1}^{n_{panels}} \int_{P_j} \mathbf{G}(\mathbf{x}_k, \mathbf{x}) \mathbf{F}_{:,j} dA(\mathbf{x}) = -8\pi\mu \mathbf{u}(\mathbf{x}_k) = -8\pi\mu \mathbf{U}_{:,k}, \tag{2.22}$$

for $k = 1 \ldots n_{\text{panels}}$, where $\mathbf{x}_k$ is the centroid of the k^{th} panel, $\mathbf{F}_{:,j}$ denotes the vector force on the jth panel and $\mathbf{U}_{:,k}$ denotes the velocity at the centroid of the kth panel. The resulting

$3n_{\text{panels}} \times 3n_{\text{panels}}$ system of equations can be represented by

$$\mathbf{G}\mathbf{F} = -8\pi\mathbf{U}. \tag{2.23}$$

Since the substrate surface S_{wall} spans a large area, discretizing (2.14) using (2.22) would require a large number of panels and would greatly increase the time and memory required to compute drag forces. Moreover, as the distance between the substrate and the suspended structures is reduced, the discretization for both S_{obj} and S_{wall} must be refined because the forces on the substrate and the bottom of the structures exhibit sharper features that require finer discretization. Therefore, discretizing the substrate greatly increases the number of unknowns in the problem, implying that only small to medium complexity problems can be solved using (2.14) with (2.22). By using a Green's function that implicitly represents the no-slip no-penetration substrate boundary condition, we remove the need to explicitly represent the substrate in (2.22) and greatly reduce the number of unknowns in $\mathbf{F}$.

Calculating the panel integrals in (2.22) requires some care because the Green's function is singular. Algorithms for calculating the panel integrals in (2.22) can be obtained by generalizing the results in [31] and are presented in Chapter 3.

Equation (2.23) is usually solved using iterative methods such as GMRES [32], and such methods compute solution approximates by forming products of $\mathbf{G}$ with candidate vectors. For discretized versions of (2.14), the matrix vector products can be computed rapidly using sparsification techniques such as multipole algorithms [33, 16, 30] or precorrected FFT (pFFT) methods [15, 34, 35]. Using pFFT methods to solve (2.19) has complications as described in Chapter 4.

Chapter 3

Panel integration

The entries in the boundary element method matrix (2.23) relate the force distribution over a flat panel P to the velocity at a test point or the weighted integral of the velocity over a test panel. The velocity due to a force distribution on a panel can be computed by integrating the Stokes Green's function. In free space, the Stokes Green's function is called the *stokeslet* and is given by (2.8); in the presence of a substrate, the Stokes Green's function is given by (2.16). In either case, the integrals of (2.8) or (2.16) can be calculated by combining the appropriate values of

$$S(m,n,p,q) = \int_P \frac{(x_0-x)^m(y_0-y)^n(z_0-z)^p}{((x_0-x)^2+(y_0-y)^2+(z_0-z)^2)^{q+\frac{1}{2}}}\mathrm{dS}. \tag{3.1}$$

In fact, combining the appropriate values of $S(m,n,p,q)$ can be used to calculate the velocity field due to any polynomial force distribution over P.

The calculation of panel integrals can be performed using analytical or numerical integration or by combining analytical and numerical integration. Analytical panel integration algorithms have the advantage that they are accurate but also that, for vector functions, much of the setup cost and the more expensive function evaluations can be reused for the different scalar entries of the vector function. On the other hand, using efficient adaptive quadrature methods for calculating the integral of vector functions has problems because each entry of the vector kernel may converge at a different rate. If the entries of the vector kernel are integrated separately many, often not trivial, calculations must be repeated. On the other hand, if the quadrature rule is applied to the vector kernel integral as a whole, the kernel with the worst convergence will determine the number of quadrature points to be used, which will also be inefficient. However, especially when the evaluation point is not close to the source panel and a fixed quadrature rule can be used for all the kernel entries, using numerical integration is often more efficient than using the analytical panel integration methods (see [36] for a list of efficient quadrature rules on triangular panels and other shapes). Our implementation uses analytical integration for the calculation of the velocity at a point due to a force distribution on a panel if the point is *close* to the panel and uses numerical quadrature [36] if the point is further away from the panel. When using Galerkin testing our implementation computes the integral over the test panel using quadrature.

In this chapter an analytical panel integration for calculating (3.1) is presented. The algorithm extends some of the results in [31], [37], [38], [39] and [40], uses some different recursion schemes and is designed for the simultaneous calculation of multiple entries of (3.1), which is particularly useful when dealing with vector kernels such as (2.8) and (2.16). The panel integration algorithm can calculate the integral of polynomial distributions over any odd power of the distance between the evaluation point and the source panel.

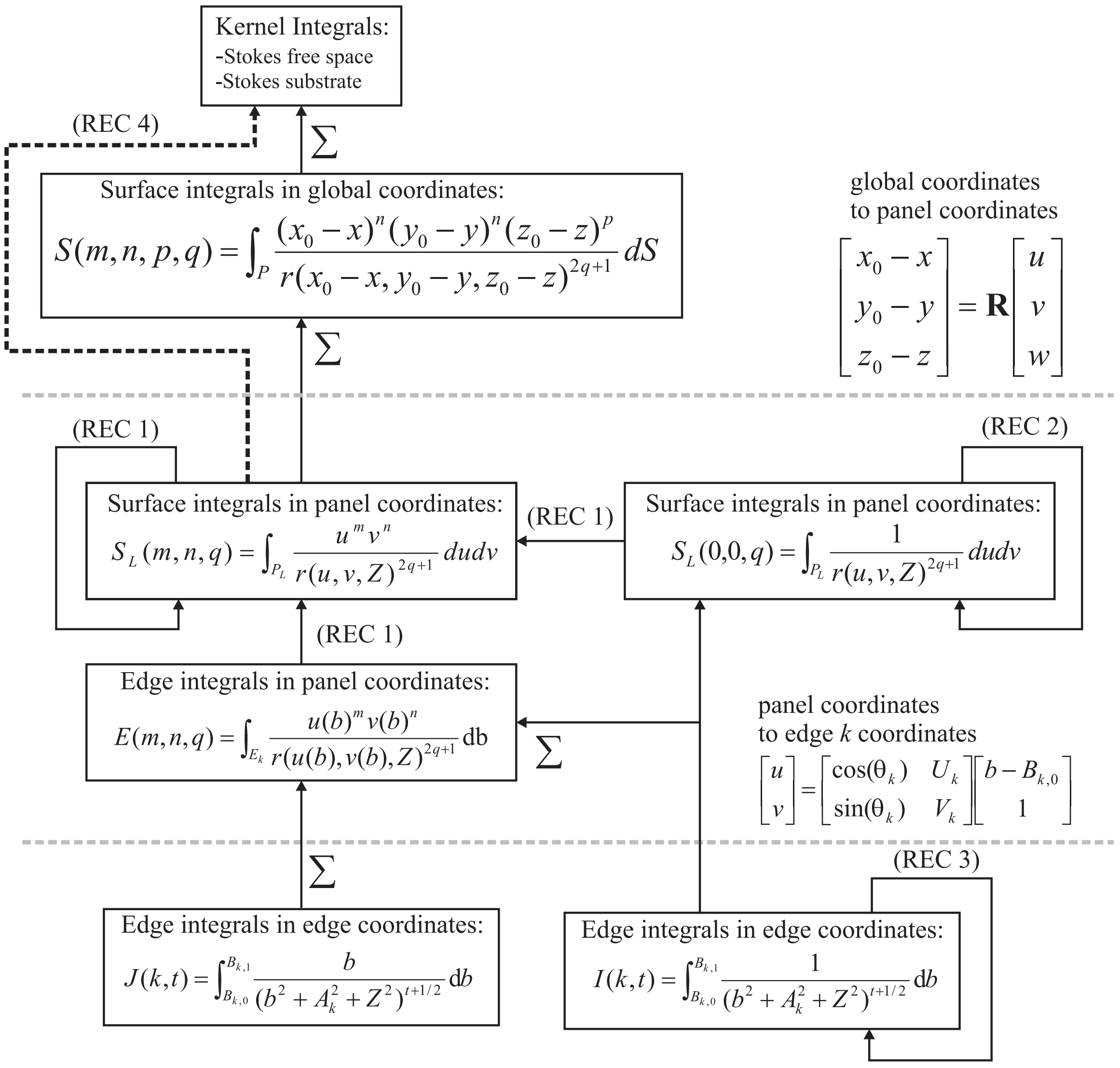
This chapter is structured as follows: first, in Section 3.1 the analytical panel integration is presented; in Section 3.2 and Section 3.3 an efficient way of assembling the Stokes Green's functions is presented; in Section 3.4 the panel integration algorithm is demonstrated; finally, in Section 3.5 some comments and suggestions for future work are made.

3.1 Analytical panel integration

In this section, an analytic method for computing (3.1) is presented. Since the algorithm relies on several recursion relations and geometric transformations an overview of the algorithm is presented in Figure 3-1.

In the following three coordinate systems will be referred to: the global coordinate system, with (x, y, z) Cartesian coordinates; the panel plane coordinate system, with coordinates

(u, v, w) defined such that the panel lies on a constant w plane; the edge k coordinate system in which the position of the target point is described as a distance b along edge k, $\mathbf{E}_k$, and a distance A_k normal to edge k in the plane of the panel. The three coordinate systems are illustrated in Figure 3-2, Figure 3-3 and Figure 3-4.



Kernel Integrals:
-Stokes free space
-Stokes substrate
(REC 4)
Σ
Surface integrals in global coordinates:
$S(m,n,p,q) = \int_P \frac{(x_0-x)^n (y_0-y)^n (z_0-z)^p}{r(x_0-x, y_0-y, z_0-z)^{2q+1}} dS$
global coordinates to panel coordinates
$\begin{bmatrix} x_0-x \\ y_0-y \\ z_0-z \end{bmatrix} = \mathbf{R} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$
(REC 1)
(REC 2)
Surface integrals in panel coordinates:
$S_L(m,n,q) = \int_{P_L} \frac{u^m v^n}{r(u,v,Z)^{2q+1}} dudv$
Surface integrals in panel coordinates:
$S_L(0,0,q) = \int_{P_L} \frac{1}{r(u,v,Z)^{2q+1}} dudv$
Edge integrals in panel coordinates:
$E(m,n,q) = \int_{E_k} \frac{u(b)^m v(b)^n}{r(u(b),v(b),Z)^{2q+1}} \mathrm{d}b$
panel coordinates to edge k coordinates
$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \cos(\theta_k) & U_k \\ \sin(\theta_k) & V_k \end{bmatrix} \begin{bmatrix} b - B_{k,0} \\ 1 \end{bmatrix}$
(REC 3)
Edge integrals in edge coordinates:
$J(k,t) = \int_{B_{k,0}}^{B_{k,1}} \frac{b}{(b^2 + A_k^2 + Z^2)^{t+1/2}} \mathrm{d}b$
Edge integrals in edge coordinates:
$I(k,t) = \int_{B_{k,0}}^{B_{k,1}} \frac{1}{(b^2 + A_k^2 + Z^2)^{t+1/2}} \mathrm{d}b$

Figure 3-1: *Overview version of the panel integration algorithm for polynomials over odd powers of the distance between the source and the target. In the graphic (REC1) refers to equation (3.4), (REC2) refers to equations (3.9) (3.10) and (REC3) refers to equation (3.11). (REC 4) corresponds to the material presented in Section 3.2 and Section 3.3.*

First a rigid body transformation from the (x, y, z) global coordinate system, illustrated in Figure 3-2, to a coordinate system (u, v, w) where the source panel is on a plane with constant w, as illustrated in Figure 3-3, is computed. In the new coordinate system the

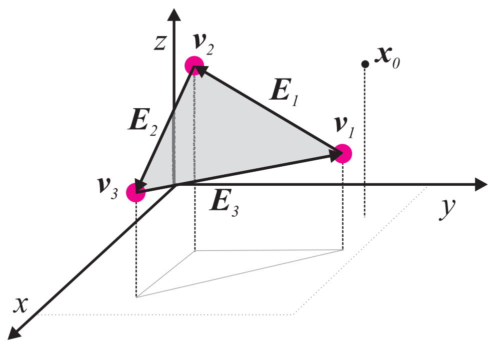


Figure 3-2: *Source panel and evaluation point in global coordinate system.*

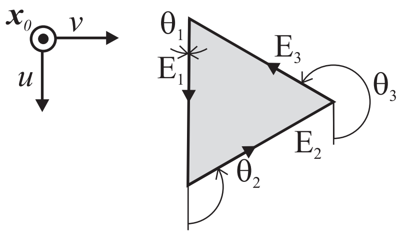


Figure 3-3: *Source panel and evaluation point in panel coordinate system.*

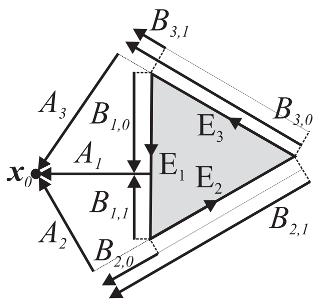


Figure 3-4: *Source panel and evaluation point in the edge coordinate systems.*

distance along w between any point in the source and the evaluation point is Z.

Considering the panel vertices and edges, as illustrated in Figure 3-2, a rotation matrix from the panel coordinate system to the shifted global coordinate system can be determine using

$$\begin{aligned}
\mathbf{R}_u &= \mathbf{E}_1/\|\mathbf{E}_1\|_2 \\
\hat{\mathbf{R}}_v &= -(\mathbf{I} - \mathbf{R}_u\mathbf{R}_u^T)\mathbf{E}_N \\
\mathbf{R}_v &= \hat{\mathbf{R}}_v/\|\hat{\mathbf{R}}_v\|_2 \\
\mathbf{R}_w &= \mathbf{R}_u \times \mathbf{R}_v
\end{aligned} \tag{3.2}$$

which can be represented in matrix form as

$$\begin{bmatrix} x_0 - x \\ y_0 - y \\ z_0 - z \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{R}_u & \mathbf{R}_v & \mathbf{R}_w \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3.3}$$

For points on the source panel, in the (u, v, w) coordinate system, w is constant and has a value of Z. Since w is constant, the integral over the surface of the source panel can be described as an integral in u and v. For a given q, the integral $S(m, n, p, q)$ can be expressed as a linear combination of integrals of $u^{m'}v^{n'}/r(u, v)^{2q+1}$ over the source

$$S_L(m', n', q) = \int_S \frac{u^{m'}v^{n'}}{r(u, v)^{2q+1}} dS$$

where $m' + n' \leq m + n + p$. However, whenever possible, it is more efficient to bypass the explicit calculation of $S(m, n, p, q)$ and to work with $S_L(m', n', q)$ instead (see Sections 3.2 and Section 3.3 for further details). In the following the prime notation is dropped for simplicity. The 2D surface integral of $u^m v^n/r(u, v)^{2q+1}$ for nonzero m or n can be computed using the divergence theorem on the (u, v) plane, which yields the recurrence relations

$$S_L(m, n, q) = \tfrac{1}{1-2q} \times \begin{cases} E_v(m, n-1, q-1) - (n-1)S_L(m, n-2, q-1) \\ E_u(m-1, n, q-1) - (m-1)S_L(m-2, n, q-1) \end{cases} \tag{3.4}$$

where E_u and E_v are the line integrals

$$\begin{bmatrix} E_u(m, n, q) \\ E_v(m, n, q) \end{bmatrix} = \sum_{k}^{n_{\text{edges}}} \begin{bmatrix} \sin\theta_k \\ -\cos\theta_k \end{bmatrix} \int_{E_k} \frac{u(l)^m v(l)^n}{r(u(l), v(l))^{2q+1}} \mathrm{d}l \tag{3.5}$$

where $[\sin\theta_k, -\cos\theta_k]$ is the k^{th} edge exterior normal in the (u, v) plane. The angles θ_k are illustrated in Figure 3-3.

Calculating $S_L(0,0,q)$ and $S_L(m,n,q)$

The values of $S_L(0,0,q)$ and $S_L(m,n,0)$ are required to initiate the recursion (3.4). To calculate $S_L(m,n,0)$, the two equations in (3.4) can be combined yielding

$$\begin{aligned} S_L(m,n,q) = \tfrac{1}{(m+n+2)-(2q+1)} \times (E_u(m+1,n,q)+ \\ E_v(m,n+1,q) - Z^2 S_L(m,n,q+1)), \end{aligned} \tag{3.6}$$

which can be used to compute $S_L(m,n,0)$ as the values $S_L(m,n,1)$ are being generated.

The value of $S_L(0,0,q)$ can be determined using cylindrical coordinates (ρ,ϕ,w) such that $u=\rho\cos\phi$ and $v=\rho\sin\phi$ and w is unchanged

$$\begin{aligned} S_L(0,0,q) = \int_{\phi_{\min}}^{\phi_{\max}} \int_{\rho_{\min}(\phi)}^{\rho_{\max}(\phi)} \frac{\rho \mathrm{d}\rho \mathrm{d}\phi}{(\rho^2+Z^2)^{q+1/2}} = \\ \frac{1}{1-2q} \int_{\phi_{\min}}^{\phi_{\max}} \frac{1}{(\rho^2+Z^2)^{q-1/2}} \bigg|_{\rho_{\min}(\phi)}^{\rho_{\max}(\phi)} \mathrm{d}\phi \end{aligned} \tag{3.7}$$

and then replacing the integrals over ϕ by integrals over the position along the edge, b,

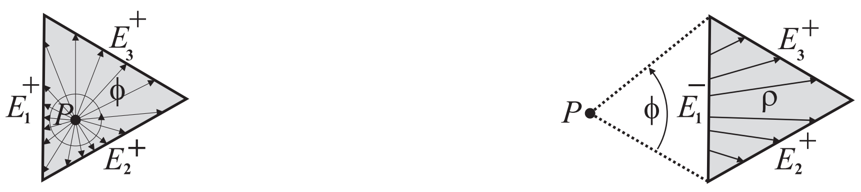
$$S_L(0,0,q) = \frac{1}{1-2q} \sum_{k=1}^{n_{\text{edges}}} \int_{B_{k,0}}^{B_{k,1}} \frac{1}{(\rho^2+Z^2)^{q-1/2}} \bigg|_{\rho_0}^{\sqrt{b^2+A_k^2}} \frac{\mathrm{d}\phi(b)}{\mathrm{d}b} \mathrm{d}b \tag{3.8}$$

where $\mathrm{d}\phi/\mathrm{d}b = A_k/(b^2+A_k)$. More explicitly

$$S_L(0,0,q) = \frac{1}{1-2q} \sum_{k=1}^{n_{\text{edges}}} A_k K(k,q) - \frac{Z^{1-2q}\Delta\phi}{1-2q} \tag{3.9}$$

where $K(k,q)$ is given by

$$K(k,q) = \int_{B_{k,0}}^{B_{k,1}} \frac{\mathrm{d}b}{(b^2+A_k^2)(b^2+A_k^2+Z^2)^{q-1/2}} = \begin{cases} A_k I(k,0) + Z\tan^{-1} \frac{Zb}{A_k r_k(b)} \Big|_{B_{k,0}}^{B_{k,1}}, & q=0 \\ -Z^{-1}\tan^{-1} \frac{Zb}{A_k r_k(b)} \Big|_{B_{k,0}}^{B_{k,1}}, & q=1 \\ \frac{A_k I(k,q-1) + (2q-3)K(k,q-1)}{(2q-1)Z^2}, & \text{otherwise} \end{cases} \tag{3.10}$$



a) When point is over panel $\Delta\phi = 2\pi$. b) When point is not over panel $\Delta\phi = 0$.

Figure 3-5: *Cylindrical coordinate system used to evaluate $S(0,0,q)$. The signs associated with each edge integral are indicated in superscript.*

where $I(k,p)$ is the edge integral calculated in the kth edge coordinate system (see Figure 3-4,

$$I(k,p) = \int_{B_{k,0}}^{B_{k,1}} \frac{\mathrm{d}b}{r_k(b)^{2p+1}} \times \begin{cases} \text{sign}\,(B_{k,0}) \log(B_{k,1}/B_{k,0}), & C_k = 0, p = 0 \\ \text{sign}\,(B_{k,0})(B_{k,1}^{2p} - B_{k,0}^{2p})/2p, & C_k = 0, p \neq 0 \\ \text{sign}\,(B_{k,0})\, \log(b + r_k(b))\big|_{B_{k,0}}^{B_{k,1}}, & C_k \neq 0, p = 0 \\ \frac{1}{2p-1}\frac{1}{C_k^2}\left(\frac{b}{r_k(b)^{2p-1}}\Big|_{B_{k,0}}^{B_{k,1}} + 2(p-1)I(k,p-1)\right), & \text{ow.} \end{cases} \tag{3.11}$$

where $r_k^2(b) = b^2 + C_k^2$ and $C_k^2 = A_k^2 + Z^2$. The expressions for $K(k,0)$ and $K(k,1)$ can be rearranged for accuracy and reduced cost, see [31] for details. To avoid loss of accuracy in (3.11), expressions of the form $a^m - b^m$ can be computed using

$$a^m - b^m = b^m \left(\frac{a}{b} - 1\right) \sum_{k=0}^{m-1} \left(\frac{a}{b}\right)^k \tag{3.12}$$

where it is assumed that $|a| < |b|$, if that is not the case a and b can simply be swapped.

The term $\Delta\phi$ in (3.9) is 2π if the evaluation point $\mathbf{x}_0$ is over the source, as illustrated in Figure 3-5a). If the evaluation point is over an edge $\Delta\phi$ is π. If $\mathbf{x}_0$ is over a corner, $\Delta\phi$ is the internal angle between the two edges that define that corner. If $\mathbf{x}_0$ is not over the panel, as illustrated in Figure 3-5b), $\Delta\phi$ is zero. If $\mathbf{x}_0$ is *on* the panel, then $S_L(0,0,q)$ is not defined for $q > 0$.

Calculating E_v and E_u

One way to calculate (3.5) would be to express u and v on each edge as a function of the position along the edge b, see Figure 3-4, using

$$u_k(b) = U_k + (b - B_{\mathrm{k},0}) \cos\theta_k = \alpha_{k,u,0} + \alpha_{k,u,1} b$$
$$v_k(b) = V_k + (b - B_{\mathrm{k},0}) \sin\theta_k = \alpha_{k,v,0} + \alpha_{k,v,1} b$$

and to combine integrals of powers of b over $r_k(b)^{2q+1}$ as in

$$\int_{E_k} \frac{u(b)^m v(b)^n}{\sqrt{b^2 + C_k^2}^{2q+1}} db = \sum_{p=0}^{m+n} \eta_{q,m,n,k,p} \int_{E_k} \frac{b^p}{\sqrt{b^2 + C_k^2}^{2q+1}} db.$$

Instead, we introduce a simpler recursion that reuses $I(k, q)$ from (3.11), which is already used to compute $S(0, 0, q)$, and

$$J(k,p) = \int_{B_{k,0}}^{B_{k,1}} \frac{b\mathrm{d}b}{(b^2 + A_k^2 + Z^2)^{p+1/2}} = \left. \frac{r_k(b)^{1-2p}}{1-2p} \right|_{B_{k,0}}^{B_{k,1}} \tag{3.13}$$

that has a simple integrand for any q. Note that, for accuracy, (3.13) can be calculated using (3.12). To appropriately combine the values of $I(k, p)$ and $J(k, p)$ we present the following recurrence relation: If

$$\underbrace{\int_{E_k} \frac{u(b)^m v(b)^n}{\sqrt{b^2 + C_k^2}^{2q+1}} db}_{E(q,m,n,k)} = \sum_p \beta_{q,m,n,k,p} \underbrace{\int_{E_k} \frac{1}{\sqrt{b^2 + C_k^2}^{2(q-p)+1}} db}_{I(q-p,k)} + \sum_p \gamma_{q,m,n,k,p} \underbrace{\int_{E_k} \frac{b}{\sqrt{b^2 + C_k^2}^{2(q-p)+1}} db}_{J(q-p,k)}$$

where $\beta_{q,0,0,k,p} = \delta_{q,p}$ and $\gamma_{q,0,0,k,p} = 0$, then

$$\begin{aligned}E(q, m+1, n, k) =& \sum_p \beta_{q,m,n,k,p} \int_{E_k} \frac{\alpha_{k,u,0} + b\alpha_{k,u,1}}{\sqrt{b^2 + C_k^2}^{2(q-p)+1}} db + \\ & \sum_p \gamma_{q,m,n,k,p} \int_{E_k} \frac{b\alpha_{k,u,0} + (b^2 + C)\alpha_{k,u,1} - C_k^2 \alpha_{k,u,1}}{\sqrt{b^2 + C_k^2}^{2(q-p)+1}} db \\ =& \sum_p \gamma_{q,m,n,k,p} \alpha_{k,u,1} I(q-p-1, k) + \\ & \sum_p [\beta_{q,m,n,k,p}\alpha_{k,u,1} + \gamma_{q,m,n,k,p}\alpha_{k,u,0}] J(q-p, k) + \\ & \sum_p [\beta_{q,m,n,k,p}\alpha_{k,u,0} - C_k^2 \gamma_{q,m,n,k,p}\alpha_{k,u,1}] I(q-p, k)\end{aligned}$$

i.e.

$$\begin{aligned}\beta_{q,m+1,n,k,p} &= \beta_{q,m,n,k,p}\alpha_{k,u,0} - C_k^2 \gamma_{q,m,n,k,p}\alpha_{k,u,1} + \gamma_{q,m,n,k,p-1}\alpha_{k,u,1} \\ \gamma_{q,m+1,n,k,p} &= \beta_{q,m,n,k,p}\alpha_{k,u,1} + \gamma_{q,m,n,k,p}\alpha_{k,u,0}\end{aligned}$$

and similarly

$$\begin{aligned}\beta_{q,m,n+1,k,p} &= \beta_{q,m,n,k,p}\alpha_{k,v,0} - C_k^2 \gamma_{q,m,n,k,p}\alpha_{k,v,1} + \gamma_{q,m,n,k,p-1}\alpha_{k,v,1} \\ \gamma_{q,m,n+1,k,p} &= \beta_{q,m,n,k,p}\alpha_{k,v,1} + \gamma_{q,m,n,k,p}\alpha_{k,v,0}\end{aligned}$$

3.2 Assembling the Stokes free space kernel integral

Fortunately, to calculate the panel integral of the Stokes free space Green's function

$$\mathbf{G}^F(\mathbf{r}) = \frac{1}{r}(\mathbf{I} + \hat{\mathbf{r}}\hat{\mathbf{r}}^T) \tag{3.14}$$

it is not necessary to explicitly calculate $S(m, n, p, q)$. Let R represent the rotation matrix associated with the source panel P and $\mathbf{r}_L = [uvw]$ the value of $\mathbf{r}$ in local panel coordinates such that $\mathbf{r} = \mathbf{R}\mathbf{r}_L$ it is clear that

$$\mathbf{G}^F(\mathbf{r}) = \frac{1}{r}(\mathbf{I} + \hat{\mathbf{r}}\hat{\mathbf{r}}^T) = \frac{1}{r}\mathbf{I} + \frac{1}{r^3}\mathbf{R}\mathbf{r}_L\mathbf{r}_L^T\mathbf{R}^T = \mathbf{R}\mathbf{G}^F(\mathbf{r}_L)\mathbf{R}^T. \tag{3.15}$$

Since $\mathbf{R}$ and $\mathbf{R}^T$ are constants, (3.15) can be integrated using,

$$\int_P \mathbf{G}^F(\mathbf{r})\mathrm{dS} = \mathbf{R}\int_{P_L} \mathbf{G}^F(\mathbf{r}_L)\mathrm{dS}_L\mathbf{R}^T \tag{3.16}$$

where P_L and dS_L indicate an integration in the panel coordinate system.

From (3.16) one can conclude that the panel integral can be calculated in panel coordinates using $S_L(m,n,q)Z^p$ and afterward surrounded by the appropriate rotations. The panel integral in local coordinates is given by

$$\int_{P_L} \mathbf{G}_L^F\mathrm{dS}_L = S_L(0,0,0)\mathbf{1}_3 + \begin{bmatrix} S_L(0,2,1) & S_L(1,1,1) & S_L(1,0,1)Z \\ S_L(1,1,1) & S_L(0,2,1) & S_L(0,1,1)Z \\ S_L(1,0,1)Z & S_L(0,1,1)Z & S_L(0,0,1)Z^2 \end{bmatrix} \tag{3.17}$$

where $\mathbf{1}_3$ represents the identity matrix with 3 rows and columns.

3.3 Assembling the Stokes substrate kernel integral

The Stokes substrate Green's function

$$\mathbf{G}^{\mathrm{S}}(\mathbf{x},\mathbf{x}_s) = \mathbf{G}^{\mathrm{F}}(\mathbf{r}) - \mathbf{G}^{\mathrm{F}}(\mathbf{r}_i) - 2hk\mathbf{G}^{\mathrm{D}}(\mathbf{r}_i)\mathbf{N} + 2h\mathbf{G}^{\mathrm{R}}(\mathbf{r}_i)\mathbf{N} \tag{3.18}$$

where

$$\mathbf{G}^{\mathrm{D}}(\mathbf{r}_i) = \frac{1}{r^3}\mathbf{I} - \frac{3}{r_i^5}\mathbf{r}_i\mathbf{r}_i^T \tag{3.19}$$

and

$$\mathbf{G}^{\mathrm{R}}(\mathbf{r}_i) = \frac{\mathbf{r}_i}{r_i^3}\mathbf{n}_w^T - \mathbf{n}_w\frac{\mathbf{r}_i}{r_i^3}^T \tag{3.20}$$

is more complicated than $\mathbf{G}^F$. The panel integral of $\mathbf{G}^{\mathrm{S}}$ can be calculated by setting up an image panel P_i, as illustrated in Figure 3-6, over which $\mathbf{G}^F(\mathbf{r}_i)$, $2hk\mathbf{G}^{\mathrm{D}}(\mathbf{r}_i)\mathbf{N}$ and $2h\mathbf{G}^{\mathrm{R}}(\mathbf{r}_i)\mathbf{N}$ can be integrated. The panel integral for the direct and image $\mathbf{G}^F$ terms can be computed by following the procedure outlined in Section 3.3.

The panel integrals for $\mathbf{G}^D$ and $\mathbf{G}^R$ present some further challenges as, form (3.18), the

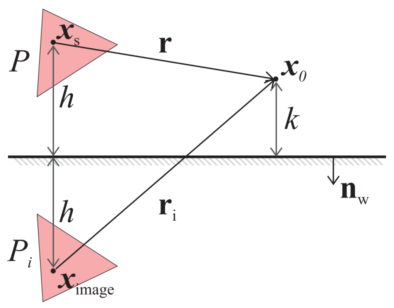


Figure 3-6: *Schematic illustration of source panel, image panel and evaluation point.*

$\mathbf{G}^D$ and $\mathbf{G}^R$ terms are scaled by a term proportional to h, the distance of the source panel to the substrate. To introduce this dependency on the distance of the source panel to the substrate, while integrating over the image panel, the identity $\mathbf{r}_i^T \cdot \mathbf{n}_w = -h - k$ can be used. Note that k is constant but h changes along the panel. Let $\mathbf{R}_i$ represent the rotation matrix associated with the image panel P_i such that $\mathbf{r}_i = \mathbf{R}_i \mathbf{r}_{i,L}$ and $\mathbf{n}_w = \mathbf{R}_i \mathbf{n}_{w,L}$, the value of h can be expressed in terms of $\mathbf{r}_{i,L}$ and $\mathbf{n}_{w,L}$ using $h = -\mathbf{r}_{i,L}^T \mathbf{n}_{w,L} - k$.

To represent $2hk\mathbf{G}^{\mathrm{D}}(\mathbf{r}_i)\mathbf{N}$ in terms of $\mathbf{r}_{i,L}$ we can use

$$\begin{aligned}2hk\mathbf{G}^{\mathrm{D}}(\mathbf{r}_i)\mathbf{N} &= -2k(\mathbf{r}_{i,L}^T\mathbf{n}_{w,L}+k)\mathbf{R}_i k\mathbf{G}^{\mathrm{D}}(\mathbf{r}_{i,L})\mathbf{R}_i^T\mathbf{R}_i \underbrace{(\mathbf{1}_3 - 2\mathbf{n}_{w,L}\mathbf{n}_{w,L}^T)}_{\mathbf{N}_L}\mathbf{R}_i^T \\ &= -\mathbf{R}_i(2k(\mathbf{r}_{i,L}^T\mathbf{n}_{w,L}+k)\mathbf{G}^{\mathrm{D}}(\mathbf{r}_{i,L})\mathbf{N}_L)\mathbf{R}_i^T.\end{aligned} \tag{3.21}$$

To represent $2h\mathbf{G}^{\mathrm{R}}(\mathbf{r}_i)\mathbf{N}$ in terms of $\mathbf{r}_{i,L}$ we can use

$$2h\mathbf{G}^{\mathrm{R}}(\mathbf{r}_i)\mathbf{N} = -\mathbf{R}_i(2(\mathbf{r}_{i,L}^T\mathbf{n}_{w,L}+k)\mathbf{G}^{\mathrm{R}}(\mathbf{r}_{i,L},\mathbf{n}_{w,L})\mathbf{N}_L)\mathbf{R}_i^T \tag{3.22}$$

The panel integrals of (3.21) and (3.22) can computed from $S(m,n,q)Z^p$. Note that, since these panel integrals are being calculated over the image panel ,P_i, $\mathbf{x}_0$ will never be *on* the panel and therefore there is no need to worry about the possibility of trying to calculate the integral of a hyper-singular expression at its singular point.

The terms associated with the dipole kernel can be calculated by combining a constant strength dipole $\mathbf{G}_{L,0}^D$ and a linear strength dipole $\mathbf{G}_{L,1}^D$. The panel integral of the constant strength dipole is given by

$$\int_{P_{i,L}} \mathbf{G}^D_{L,0} \mathrm{dS}_L = S_L(0,0,1)\mathbf{1}_3 - 3 \begin{bmatrix} S_L(0,2,2) & S_L(1,1,2) & S_L(1,0,2)Z \\ S_L(1,1,2) & S_L(0,2,2) & S_L(0,1,2)Z \\ S_L(1,0,2)Z & S_L(0,1,2)Z & S_L(0,0,2)Z^2 \end{bmatrix} \tag{3.23}$$

where $P_{i,L}$ represents the image panel in the image panel coordinate system.

Let

$$\alpha = [S_L(1,0,1)\ S_L(0,1,1)\ S_L(0,0,1)Z]\, \mathbf{n}_{w,L}$$

and

$$\beta_{i,j} = \sum_{k=1}^{3} S_L(\delta_{i,1}+\delta_{j,1}+\delta_{k,1}, \delta_{i,2}+\delta_{j,2}+\delta_{k,2}, 2) Z^{\delta_{i,3}+\delta_{j,3}+\delta_{k,3}} \mathbf{n}_{w,L,k}$$

where $\delta_{i,k}$ is the Kronecker delta function and $\mathbf{n}_{w,L,k}$ is the k^{th} component of the substrate normal in the image local panel coordinate system. The panel integral of the linear strength dipole is given by

$$\int_{P_{i,L}} \mathbf{G}^D_{L,1} \mathrm{dS}_L = \alpha \mathbf{1}_3 - 3 \begin{bmatrix} \beta_{1,1} & \beta_{1,2} & \beta_{1,3} \\ \beta_{2,1} & \beta_{2,2} & \beta_{2,3} \\ \beta_{3,1} & \beta_{3,2} & \beta_{3,3} \end{bmatrix} \tag{3.24}$$

Therefore, the total contribution due to the dipole terms is

$$\int_{P_{i,L}} \mathbf{G}^D_L \mathrm{dS}_L = -2k \left(\int_{P_{i,L}} \mathbf{G}^D_{L,1} \mathrm{dS}_L - k \int_{P_{i,L}} \mathbf{G}^D_{L,0} \mathrm{dS}_L \right). \tag{3.25}$$

The contributions associated with the rotlet term, $\mathbf{G}^R$, consists of the sum of the panel integral of a constant strength rotlet $\mathbf{G}^R_{L,0}$ and the panel integral of a linear strength rotlet $\mathbf{G}^R_{L,1}$ with strength $\mathbf{r}^T_{i,L}\mathbf{n}_{w,L}$. Let $\boldsymbol{\gamma}_0 = [S(1,0,1)\ S(0,1,1)\ S(0,0,1)Z]^T$, the panel integral of the constant strength rotlet is given by

$$\int_{P_{i,L}} \mathbf{G}^R_{L,0} \mathrm{dS}_L = \boldsymbol{\gamma}_0 \mathbf{n}^T_{w,L} - \mathbf{n}_{w,L} \boldsymbol{\gamma}^T_0 . \tag{3.26}$$

Let

$$\boldsymbol{\gamma}_1 = \begin{bmatrix} [S_L(2,0,1)\ S_L(1,1,1)\ S_L(1,0,1)Z]\,\mathbf{n}_{w,L} \\ [S_L(1,1,1)\ S_L(0,2,1)\ S_L(0,1,1)Z]\,\mathbf{n}_{w,L} \\ [S_L(1,0,1)Z\ S_L(0,1,1)Z\ S_L(0,0,1)Z^2]\,\mathbf{n}_{w,L} \end{bmatrix} \tag{3.27}$$

the panel integral of the linear strength rotlet is

$$\int_{P_{i,L}} \mathbf{G}^R_{L,1} \mathrm{dS}_L = \boldsymbol{\gamma}_1 \mathbf{n}^T_{w,L} - \mathbf{n}_{w,L} \boldsymbol{\gamma}^T_1 \tag{3.28}$$

and the total rotlet contribution is

$$\int_{P_{i,L}} \mathbf{G}^R_L \mathrm{dS}_L = 2\left(k \int_{P_{i,L}} \mathbf{G}^R_{L,0} \mathrm{dS}_L - \int_{P_{i,L}} \mathbf{G}^R_{L,0} \mathrm{dS}_L\right). \tag{3.29}$$

The integral of the Stokes substrate Green's function is then given by

$$\begin{aligned} \int_P \mathbf{G}^S \mathrm{dS} =& \mathbf{R} \int_{P_L} \mathbf{G}^F \mathrm{dS} \mathbf{R}^T - \mathbf{R}_i \int_{P_{i,L}} \mathbf{G}^F \mathrm{dS} \mathbf{R}^T_i + \\ & \mathbf{R}_i \left(\int_{P_{i,L}} \mathbf{G}^D_L \mathrm{dS}_L - \int_{P_{i,L}} \mathbf{G}^R_L \mathrm{dS}_L \right) \mathbf{N}_{i,L} \mathbf{R}^T_i \end{aligned} \tag{3.30}$$

Galerkin

If the boundary element method is using Galerkin testing, the entries of the boundary element matrix are the weighted integrals over a test panel of the velocity due to a force distribution over a source panel. To calculate the entries with Galerkin testing our implementation calculates the integral over the test panel using quadrature rules (see [36]).

3.4 Testing

In this section we demonstrate the panel integration algorithm by plotting the velocity field produced by a force along the x, y or z direction. The velocity field produced by a force distribution on a square panel with 1μm side at $z = 2\mu$m, parallel to the substrate, is illustrated in Figure 3-7. For comparison, where for comparison, the velocity field in the absence of the substrate is also illustrated in Figure 3-7.

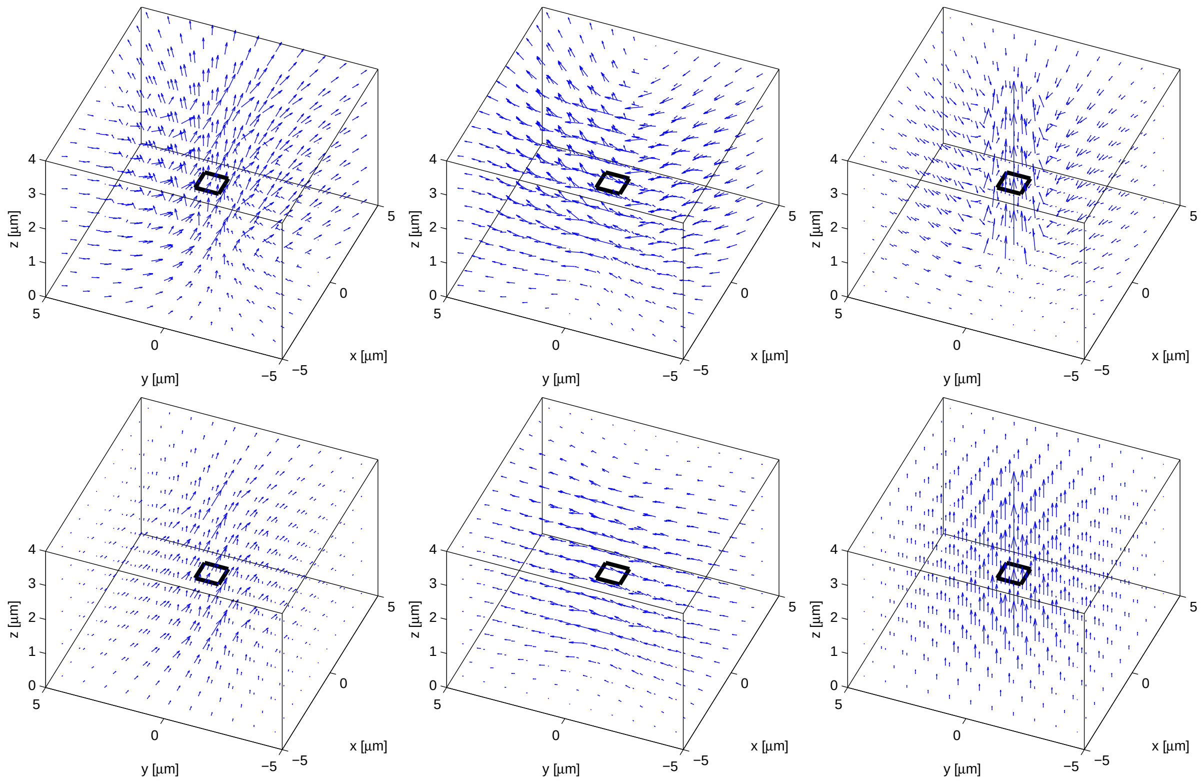


Figure 3-7: *Velocity field produced by a constant force distribution on a square panel with 1μm side at $z = 2\mu$m. The figures on the top row were produced by integrating the Stokes substrate Green's function. The figures on the bottom row were produced by integrating the Stokes free space Green's function. On the first column, a force along the x axis was applied; on the second column, a force along y was applied; on the third column, a force along z was applied. To improve visualization of the field, the length of the arrows in the figures is proportional to the logarithm of the magnitude of the velocity.*

The velocity field produced by a force distribution on a triangular panel with 1μm side at $z = 2\mu$m, normal to the substrate, is illustrated in Figure 3-7. For comparison, where for comparison, the velocity field in the absence of the substrate is also illustrated in Figure 3-7.

The panel integration algorithm was tested by comparing its results to those produced using adaptive subdivision quadrature methods using the quadrature rules in [36] as the inner quadrature rule. The results were verified to match to any reasonable degree of accuracy.

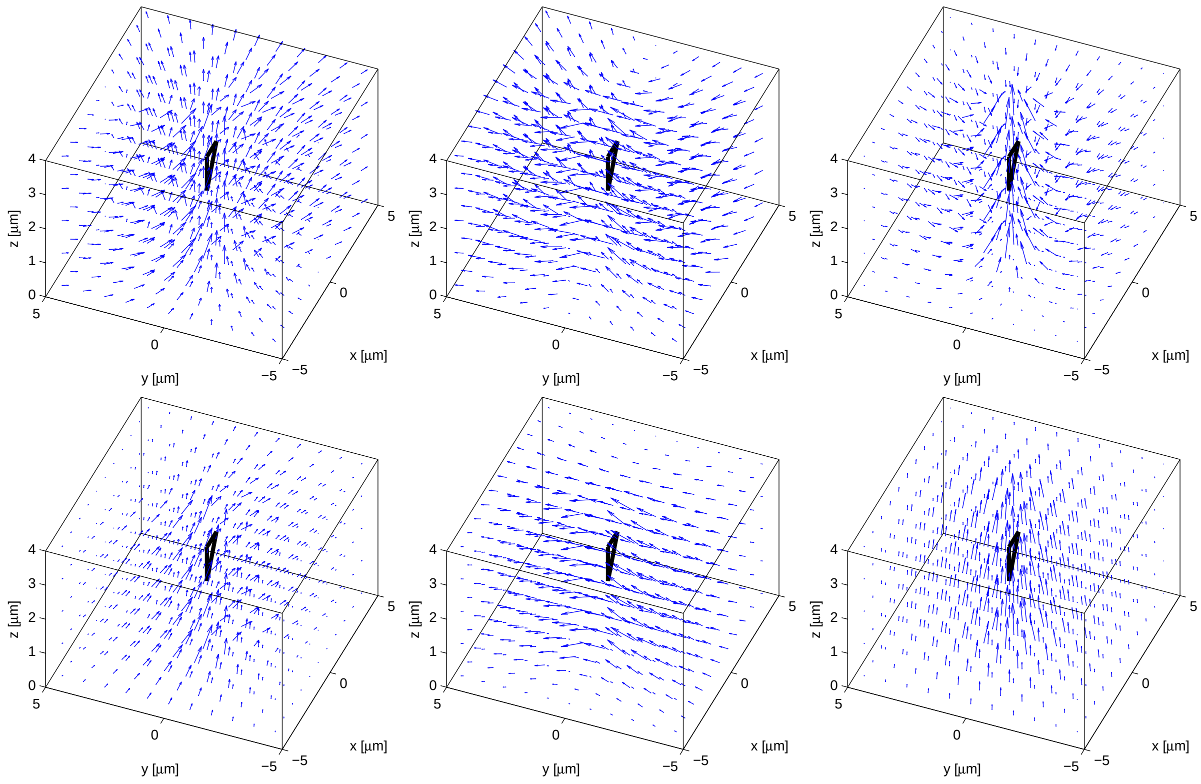


Figure 3-8: *Velocity field produced by a constant force distribution on a triangular panel with 1μm side at $z = 2\mu$m placed normal to the substrate. The figures on the top row were produced by integrating the Stokes substrate Green's function. The figures on the bottom row were produced by integrating the Stokes free space Green's function. On the first column, a force along the x axis was applied; on the second column, a force along y was applied; on the third column, a force along z was applied. To improve visualization of the field, the length of the arrows in the figures is proportional to the logarithm of the magnitude of the velocity.*

3.5 Notes

The panel integration algorithm presented in this chapter does not account for the case where the evaluation point is *on* an edge or corner of the panel. Extending the panel integration algorithm to deal with such cases is not too complicated but it was not necessary for the purposes of developing the boundary element solver because, for that application, the evaluation points where either not on the source panel or were interior points on the source panel.

The velocity due a linear strength force distribution can be calculated by replacing $S_L(m, n, q)$

in Sections 3.2 and 3.3 by

$$\hat{S}_L(m,n,q) = \begin{bmatrix} S_L(m,n,q) \\ S_L(m+1,n,q) \\ S_L(m,n+1,q) \end{bmatrix} \tag{3.31}$$

and replacing scalar operations by the corresponding pointwise vectorized operations. Naturally the same process can be applied to the computation of the velocity due to higher order force distributions.

Chapter 4

Precorrected FFT solver for Stokes flow

In this chapter we describe an accelerated boundary element solver for calculating the drag force on microelectromechanical and microfluidic devices. The drag force on microelectromechanical devices such as oscillators, accelerometers, combdrives and micromirrors is an important factor that significantly influence their dynamic behavior [3, 4, 5]. Especially for surface micromachined devices, the drag is greatly influenced by the presence of the nearby substrate [7]. Stokes drag near the bottom of a microfluidic channel is also important for the calculation of the cell trapping dynamics of structures such as those described in [21], [22] and [23]. However, explicitly accounting for the substrate can be computationally expensive.

Accelerated boundary element solvers based on the multipole method [13], panel clustering and wavelets [14] and on the precorrected FFT method [15], have previously been applied to the calculation of drag forces on MEMS structures [6, 16, 17, 18, 19]. However, with the exception of the variable order wavelet method used in [19], these implementations of the boundary element method use a free-space Green's function that requires an explicit discretization of both the substrate and the suspended structures. The problem with discretizing the substrate is that a large number of unknowns are needed. In addition, when the structure is close to the substrate, the substrate discretization must be refined to match the "shadow" of the structure.

In this chapter we present a more efficient fast BEM solver for structures above a substrate that uses the Green's function for Stokes flow bounded by a plane to implicitly represent the device substrate. In particular, we develop an approach based on the precorrected FFT (pFFT) algorithm [15], as the pFFT method has been demonstrated to be extremely memory efficient. Such pFFT approaches rely on translation invariance and, as we describe below, that introduces algorithmic complications when combined with the substrate Stokes Green's function.

In the following, it is shown that using the Stokes substrate Green's function significantly reduces memory usage and the time required to calculate the drag force. It is also shown that, if the substrate is represented explicitly, then for small separation distances between the structures and the substrate a very large number of panels must be used to represent the substrate. In Chapter 5, it is also demonstrated that to achieve a given level of accuracy, regardless of whether an implicit or an explicit representation of the substrate is used, the size of the panels used to discretize the structures must be reduced as the distance between the structures and the substrate decreases. This result is surprising because the need to refine the discretization of the structures as the gap decreases is not necessarily driven by a need to more accurately represent the solution. Nevertheless, it is demonstrated that, despite the complications introduced in the pFFT algorithm and the fact that the discretization still needs to be refined as the structures are brought closer to the substrate, using the Stokes flow substrate Green's function is still worthwhile as it produces more accurate results more efficiently than by using an explicit substrate discretization.

This chapter is structured as follows. In Section 4.1, the pFFT algorithm is reviewed. In Section 4.2, the modifications to the pFFT algorithm required to support the Stokes substrate Green's function are presented. In Section 4.3, results validating and demonstrating the pFFT accelerated boundary element method using the substrate Green's function are shown. Finally, in Section 4.4, the advantages and limitations of the approach proposed in this chapter are discussed. The background material, associated with the Stokes flow model and boundary integral formulation is presented in Chapter 2.

4.1 Precorrected FFT algorithm

The precorrected FFT algorithm, introduced in [15], accelerates the process of calculating the interactions between N sources and M targets by first applying interpolation to map the effects of the N arbitrarily located sources to a regular grid, of N_G points, then calculating the interactions between the N_G regular grid points using FFT accelerated convolution and finally interpolating the results from the grid to the M targets. Since the interactions calculated using this grid based procedure are not accurate enough when the sources and targets are nearby, in the pFFT algorithm the values of the inaccurate grid based nearby interactions are discarded (effectively subtracted from the result produced by the FFT accelerated convolution) and replaced with more accurate estimates of the nearby interactions, usually obtained with numerical or analytical integration.

To use the FFT to accelerate the convolution step on a regular grid, the interactions between sources and targets must be translation invariant. However, the substrate Green‘s function for Stokes flow is not translation invariant on the direction normal to the substrate. In the following sections the basic concepts and steps involved in the pFFT algorithm are reviewed.

4.1.1 Projection and interpolation

Consider evaluating a function $g(x_t, x_s, y_t, y_s, z_t, z_s) = g(\mathbf{x}_t, \mathbf{x}_s)$, where $\mathbf{x}_s$ denotes a source point and $\mathbf{x}_t$ denotes a target point. The function g can be approximated by polynomially interpolating from a set of samples $g(\mathbf{x}_t, \mathbf{x}_p)$ evaluated at a set of points $\mathbf{x}_p$ near the source point $\mathbf{x}_s$. In particular,

$$g(\mathbf{x}_t, \mathbf{x}_s) = \sum_p g(\mathbf{x}_t, \mathbf{x}_p) L_p(\mathbf{x}_s) + E_p(\mathbf{x}_t, \mathbf{x}_s) \tag{4.1}$$

where, for a given projection order p, $L_p(\mathbf{x}_s)$ is a Lagrangian interpolator, i.e. a polynomial that is 1 when $\mathbf{x}_s = \mathbf{x}_p$ and zero for the remaining sample points, and $E_p(\mathbf{x}_t, \mathbf{x}_s)$ is the projection error. In the context of the pFFT algorithm, this *source interpolation* is called projection.

Dually, $g(\mathbf{x}_t, \mathbf{x}_s)$ can be approximated by interpolating samples $g(\mathbf{x}_i, \mathbf{x}_s)$ evaluated at a set of points $\mathbf{x}_i$ near the target point $\mathbf{x}_t$. Analogous to (4.1),

$$g(\mathbf{x}_t, \mathbf{x}_s) = \sum_i L_i(\mathbf{x}_t) g(\mathbf{x}_i, \mathbf{x}_s) + E_i(\mathbf{x}_t, \mathbf{x}_s) \tag{4.2}$$

where $L_i(\mathbf{x}_s)$ is again a Lagrangian interpolator and $E_i(\mathbf{x}_t, \mathbf{x}_s)$ is the interpolation error.

Projecting the source using (4.1) and interpolating at the destination (4.2) results in

$$g(\mathbf{x}_t, \mathbf{x}_s) = \sum_i L_i(\mathbf{x}_t) \sum_p g(\mathbf{x}_i, \mathbf{x}_p) L_p(\mathbf{x}_s) + E_{i+p}(\mathbf{x}_t, \mathbf{x}_s) \tag{4.3}$$

where the $\mathbf{x}_i$'s and $\mathbf{x}_p$'s are conveniently chosen points (e.g. a subset of points on a uniform grid) and E_{i+p} represents the approximation error.

The accuracy of the above approximation can be improved by using additional information about g, such as its derivatives with respect to the source and field positions.

4.1.2 Collocation

The BEM collocation matrix $\mathbf{G}$ in (2.23) can be approximated using (4.3)

$$\begin{aligned} &\textstyle\sum_{\mathrm{s}} \mathbf{G}_{t,\mathrm{s}} \mathbf{f}_{\mathrm{s}} = \sum_{\mathrm{s}} \int_{S_{\mathrm{s}}} g(\mathbf{x}_t, \mathbf{x}_s) f(\mathbf{x}_s) dS = \\ &\quad \textstyle\sum_i \sum_p L_i(\mathbf{x}_t) g(\mathbf{x}_i, \mathbf{x}_p) \sum_{\mathrm{s}} \int_{S_{\mathrm{s}}} L_p(\mathbf{x}_s) f(\mathbf{x}_s) dS + \\ &\quad \textstyle\sum_{\mathrm{s}} \int_{S_{\mathrm{s}}} E_{i+p}(\mathbf{x}_{s_k}, \mathbf{x}_t) f(\mathbf{x}_{s_k}) dS = \\ &\quad \underbrace{\sum_i \mathbf{I}_{t,i} \sum_p \mathbf{g}_{i,p} \sum_{\mathrm{s}} \mathbf{P}_{p,\mathrm{s}}}_{\mathbf{G}^{\mathrm{i+p}}_{t,\mathrm{s}}} \mathbf{f}_{\mathrm{s}} + \textstyle\sum_{\mathrm{s}} \mathbf{E}_{t,\mathrm{s}} \mathbf{f}_{\mathrm{s}}, \end{aligned} \tag{4.4}$$

where the matrix $\mathbf{P}_{p,\mathrm{s}}$ is called the projection matrix, and the matrix $\mathbf{I}_{t,i}$ is called the interpolation matrix. The error term $\mathbf{E}_{t,\mathrm{s}}$ maybe large if $\mathbf{x}_{\mathrm{s}}$ and $\mathbf{x}_t$ are nearby or if the interpolation stencil for $\mathbf{x}_t$ overlaps with the projection stencil for $\mathbf{x}_{\mathrm{s}}$ and the kernel is singular, in which case the kernel values $g(\mathbf{x}_i, \mathbf{x}_p)$ for $\mathbf{x}_i = \mathbf{x}_p$, which cannot be correctly evaluated, will corrupt the approximation. In the precorrected FFT algorithm the error term for nearby interactions i.e., $\mathbf{E}_{t,\mathrm{s}} = \mathbf{G}_{t,\mathrm{s}} - \mathbf{G}_{t,\mathrm{s}}^{\mathrm{i+p}}$, is calculated explicitly and added to the contributions calculated using projection and interpolation in order to improve the accuracy of the approximation. For distant interactions, $\mathbf{E}_{t,\mathrm{s}}$ is negligible and is set to zero thus generating a sparse matrix $\hat{\mathbf{E}}_{t,\mathrm{s}}$ called the precorrection matrix.

The non-zero entries of the precorrection matrix are generated using quadrature schemes [36] or an analytical method to calculate an accurate value for $\mathbf{G}_{t,s}$ and by subtracting the grid based contribution $\mathbf{G}_{t,\mathrm{s}}^{\mathrm{i+p}}$. An analytical method for calculating the velocity field due to a force distribution on a flat panel is presented in Chapter 3.

4.1.3 Convolution on regular grid

If the points $\mathbf{x}_i$ and $\mathbf{x}_p$ are points on regularly spaced grids with the same spacing and the same axis of alignment, and $\mathbf{g}_{i,p} = g(\mathbf{x}_i, \mathbf{x}_p)$ is translation invariant[1], then $\mathbf{g}_{i,p} = \hat{\mathbf{g}}_{i-p}$ is a block Toeplitz matrix. Therefore, the term in (4.4),

$$\sum_p \mathbf{g}_{i,p} \underbrace{\sum_s \mathbf{P}_{p,s}\mathbf{f}_s}_{\hat{\mathbf{f}}_p} = \sum_p \mathbf{g}_{i,p}\hat{\mathbf{f}}_p = \sum_p \hat{\mathbf{g}}_{i-p}\hat{\mathbf{f}}_p \tag{4.5}$$

can be interpreted as a discrete convolution and can be calculated efficiently using the FFT algorithm [41]. Calculating the convolution using the FFT to transform both a zero padded $\hat{\mathbf{f}}_p$ and the kernel associated with $\hat{\mathbf{g}}_{i-p}$ to the frequency domain, performing a pointwise multiplication, and then inverse transforming the result has a computational cost of $O(N_\mathrm{G}\log(N_\mathrm{G}))$. By contrast, calculating the convolution on the grid directly has a cost of $O(N_\mathrm{G}^2)$. A similar interpretation and acceleration scheme can be used for the case where the kernel is of the form $g(\mathbf{x}_i, \mathbf{x}_p) = g(\mathbf{x}_i - D\mathbf{x}_p)$ where $\mathbf{D}$ is a 3 by 3 diagonal matrix with entries that are 1 or -1. For the grid axis corresponding to -1 entries the matrix $\mathbf{g}_{i,p}$

[1]A function is translation invariant if $g(\mathbf{x}_i, \mathbf{x}_p)$ only depends on the relative position of $\mathbf{x}_i$ and $\mathbf{x}_p$ and the two parameter function $g(\mathbf{x}_i, \mathbf{x}_p)$ can be reduced to a single parameter function $g(\mathbf{x}_i - \mathbf{x}_p)$.

forms Hankel blocks that can be viewed as a discrete convolution of an image source with a shifted translation invariant kernel [15].

In the following, for simplicity, we will assume that the grid of N_{G} points is arranged in N_{Z} regularly spaced layers with with N_{XY} points parallel to the substrate.

4.2 Dealing with the substrate Green's function

Since the Stokes flow substrate Green's function is not translation invariant along the direction normal to the substrate, the FFT based convolution described in section 4.1.3 cannot be directly applied to accelerate the calculation of the velocity on the N_{G} grid points. Examining how the Stokes flow substrate Green's function is not translation invariant along the direction normal to the substrate leads to several alternatives for extending the pFFT approach.

The Stokes flow substrate Green's function, repeated from (2.18),

$$\mathbf{G}^{\mathrm{w}}(\mathbf{x},\mathbf{x}_s)=\mathbf{G}^{\mathrm{F}}(\mathbf{r})-\underbrace{\mathbf{G}^{\mathrm{F}}(\mathbf{r}_i)-2hk\mathbf{G}^{\mathrm{D}}(\mathbf{r}_i)\mathbf{N}+2h\mathbf{G}^{\mathrm{R}}(\mathbf{r}_i)\mathbf{N}}_{\mathbf{G}^{\mathrm{im}}(\mathbf{x},\mathbf{x}_{\mathrm{i}})}$$

is not translation invariant along the direction normal to the substrate for two reasons: first, it is the combination of a direct contribution and an image contribution; second, some of the image terms are multiplied by h or hk and therefore depend on the absolute position of the source and the target. The first issue can be addressed by splitting the kernel into a direct contribution $\mathbf{G}(\mathbf{x},\mathbf{x}_0)^{\mathrm{F}}$ and an image contribution $\mathbf{G}^{\mathrm{im}}(\mathbf{x},\mathbf{x}_{\mathrm{i}})$. The direct contribution is translation invariant and does not pose any additional challenge. However, the image contribution $\mathbf{G}^{\mathrm{im}}(\mathbf{x},\mathbf{x}_{\mathrm{i}})$ has an explicit dependence on h and hk and is not translation invariant even though the kernels $\mathbf{G}^{\mathrm{F}}$, $\mathbf{G}^{\mathrm{D}}$ and $\mathbf{G}^{\mathrm{SD}}$ are. To deal with the translation variant terms, it is possible to move the h and the hk dependence to the projection and interpolation matrices as in

$$\begin{aligned}
&\mathbf{u}_t = \textstyle\sum_{\mathrm{s}} \int_{S_{\mathrm{s}}} G^{\mathrm{w}}(\mathbf{x}_t, \mathbf{x}_{\mathbf{s}}) dS_s = \sum_i L_i(\mathbf{x}_t) \times (\\
&\quad \textstyle\sum_p G^{\mathrm{F}}(\mathbf{x}_i, \mathbf{x}_p) \sum_s \int_{S_{\mathrm{s}}} L_p(\mathbf{x}) dS_s \mathbf{f}_{\mathrm{s}} - \\
&\quad \textstyle\sum_p G^{\mathrm{F}}(\mathbf{x}_i, \mathbf{N}\mathbf{x}_p) \sum_s \int_{S_{\mathrm{s}}} L_p(\mathbf{x}) dS_s \mathbf{f}_{\mathrm{s}} + \\
&\quad \textstyle\sum_p 2G^{\mathrm{R}}(\mathbf{x}_i, \mathbf{N}\mathbf{x}_p) \sum_s \int_{S_{\mathrm{s}}} L_p(\mathbf{x}) h(\mathbf{x}) dS_s \mathbf{f}_{\mathrm{s}} + \\
&\quad k(\mathbf{x}_t) \textstyle\sum_p 2G^{\mathrm{D}}(\mathbf{x}_i, \mathbf{N}\mathbf{x}_p) \sum_{\mathrm{s}} \int_{S_{\mathrm{s}}} L_p(\mathbf{x}) h(\mathbf{x}) dS_s \mathbf{f}_{\mathrm{s}}),
\end{aligned} \tag{4.6}$$

which can be written in matrix form as

$$\begin{aligned}
\mathbf{u}_t = \; & \textstyle\sum_{\mathrm{s}} \int_{S_{\mathrm{s}}} \mathbf{G}^{\mathrm{w}}(\mathbf{x}_t, \mathbf{x}_{\mathbf{s}}) \mathbf{f}_{\mathrm{s}} dS_s = \\
& \textstyle\sum_i \mathbf{I}_{t,i} \sum_p \mathbf{G}^{\mathrm{F}}_{i,p} \sum_{\mathrm{s}} \mathbf{P}_{p,\mathrm{s}} \mathbf{f}_{\mathrm{s}} - \\
& \textstyle\sum_i \mathbf{I}_{t,i} \sum_p \mathbf{G}^{\mathrm{im,F}}_{i,p} \mathbf{P}_{p,\mathrm{s}} \mathbf{f}_{\mathrm{s}} + \\
& \textstyle\sum_i \mathbf{I}_{t,i} \sum_p 2\mathbf{G}^{\mathrm{im,R}}_{i,p} \sum_{\mathrm{s}} \mathbf{P}^{(1)}_{p,\mathrm{s}} \mathbf{f}_{\mathrm{s}} + \\
& \textstyle\sum_i \mathbf{I}^{(1)}_{t,i} \sum_p 2\mathbf{G}^{\mathrm{im,D}}_{i,p} \sum_{\mathrm{s}} \mathbf{P}^{(1)}_{p,\mathrm{s}} \mathbf{f}_{\mathrm{s}}.
\end{aligned} \tag{4.7}$$

The resulting FFT accelerated matrix vector product, without the precorrection term, is illustrated in Figure 4-1. Using this scheme each matrix vector product requires 6 scalar projections, 6 FFTs, 6 iFFTs and 6 scalar interpolations. Using the symmetry of the kernels, this scheme requires storing 6 scalar kernels transforms for the direct contribution $\mathbf{G}^{\mathrm{F}}(\mathbf{x}, \mathbf{x}_0)$ and 14 scalar kernel transforms for the image contribution $\mathbf{G}^{\mathrm{im}}(\mathbf{x}, \mathbf{x}_{\mathrm{i}})$.

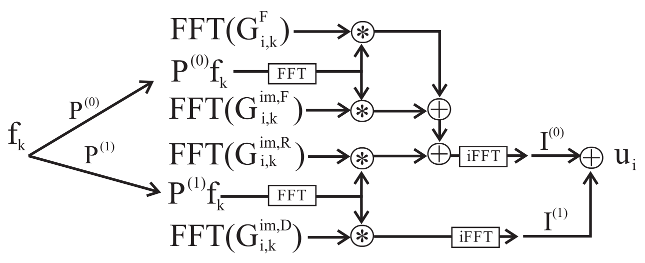


Figure 4-1: *The velocity on the interpolation grid* $\mathbf{u}$ *can be calculated using 6 projections, 6 FFTs, 6 IFFTs and 6 interpolations and uses* $30N_{\mathrm{G}}$ *normalized storage for kernel transforms. Note that the transforms of the Green's functions are computed once and stored.*

Splitting the kernel into 20 scalar components while the original kernel has 8 independent scalar entries seems wasteful. Motivated by this observation, an alternative approach, one that does not require splitting the kernel nor modifying the projection and interpolation steps, was considered. Instead of splitting the kernel and using a three dimensional FFT to accelerate the convolution, in the alternative approach the velocity on each of the N_{Z} layers of the grid is computed using an explicit convolution of the force distribution on each of the N_{Z} layers. Since G^{w} is translation invariant along directions parallel to the substrate, the alternative approach can use two dimensional FFTs to accelerate the computation of the contribution of the velocity on a layer due to the force on another layer. Therefore, the computational cost of the alternative approach is $O(N_{\mathrm{Z}}^2 N_{\mathrm{XY}} \log N_{\mathrm{XY}}) = O(N_{\mathrm{Z}} N_{\mathrm{G}} \log N_{\mathrm{XY}})$. Since G^{w} is not translation invariant in the direction normal to the substrate and there are 8 scalar kernels, to store the interactions between all the pairs of layers requires memory proportional to $8N_{\mathrm{Z}}^2 N_{\mathrm{XY}} = 8N_{\mathrm{Z}} N_{\mathrm{G}}$.

Despite its simplicity, this alternative approach was abandoned because it does not scale well as N_{z} increases. Even though splitting the kernel into subcomponents complicates the projection and interpolation steps and requires more memory, the memory used by the split kernel approach scales linearly with the number of grid points, while the memory and time required for the alternative approach grow quadratically with N_{Z}. We found that for $N_{\mathrm{Z}} > 4$, the split kernel approach was more memory efficient.

4.3 Results and discussion

In this section we compare the results obtained using the pFFT accelerated BEM formulation using the ground plane Green's function with theoretical, numerical and experimental results.

4.3.1 Sphere moving near a plane wall

The Stokes drag on a sphere moving parallel or normal to a plane wall has an analytical solution (see [42] and [43] for details). For comparison, the drag force on a sphere dis-

gap	Normalized drag - parallel motion			Normalized drag - normal motion		
h/r-1	coarse mesh	fine mesh	exact [42]	coarse mesh	fine mesh	exact [43]
15	1.0353	1.0361	1.0364	1.0742	1.0752	1.0755
7	1.0741	1.0751	1.0754	1.1608	1.1621	1.1625
3	1.1603	1.1615	1.1620	1.3772	1.3794	1.3802
1	1.3797	1.3820	1.3828	2.1163	2.1232	2.1255
0.5	1.5908	1.5945	1.5957	3.1812	3.1993	3.2054
0.2	1.9425	1.9501	1.9527	6.2314	6.3131	6.3409

Table 4.1: *Normalized drag force on a sphere of radius r whose center is at a distance h from a wall.*

cretization with 1280 panels, labeled *coarse*, and for a sphere with 5120 panels, labeled *fine*, for both in-plane and normal motion were calculated using the precorrected FFT approach described above. The drag force values, normalized with respect to the Stokes drag force in free space $-6\pi\mu rU$, where r is the sphere radius and U is its linear velocity, are summarized in Table 4.1 and illustrated in Figure 4-2. From the table and the figures, it can be seen that the agreement between the theoretical values and the solution produced by the above method is very good.

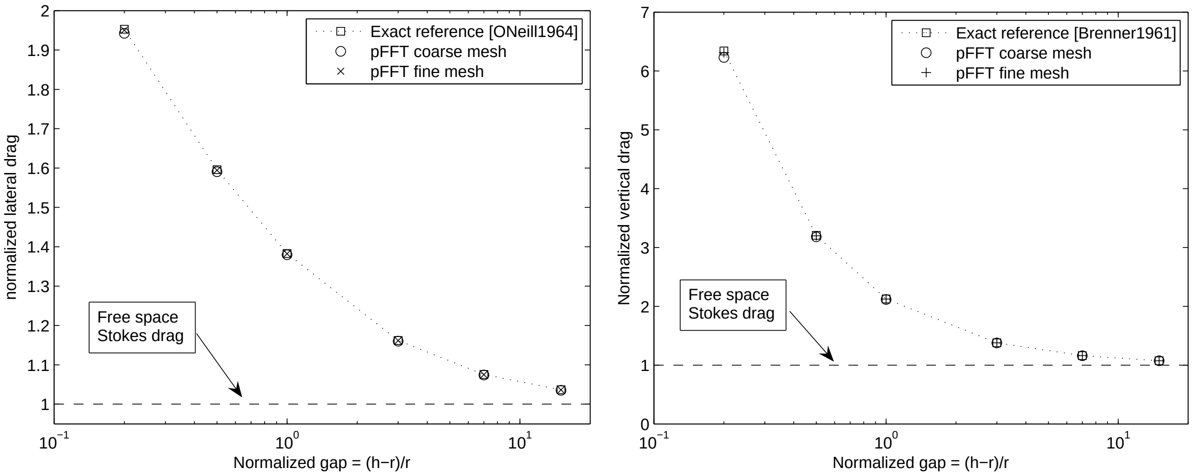


Figure 4-2: *Normalized drag force density on a sphere of radius r whose center is at a distance h from a plane wall. The drag is normalized with respect to the Stokes drag force in free space $6\pi\mu rU$. On the left is the the drag associated with motion normal to the wall, on the right is the drag associated with motion parallel to the wall.*

4.3.2 Cylinder over substrate - Effect of substrate discretization

The focus of this section is to analyze the effect of the substrate discretization on the solution accuracy for a simple structure. The example structure was chosen to be a cylinder with 10μm radius and 2μm thickness. A cylinder discretizations with a median panel radius of 0.14μm was constructed using Comsol 3.2.

The cylinder mesh was placed over a 40μm by 40μm plane discretized into a set of regular square panels. The cylinder mesh was then placed at a distance of 5μm, 2μm, 1μm and 0.5μm from the substrate. For each configuration, the drag force on the bottom of the cylinder was calculated for lateral motion of the cylinder and the total vertical force on the cylinder was calculated for vertical motion. A reference solution was generated by solving the same problem but using the implicit substrate representation.

The relative error of the lateral drag force on the bottom of the cylinder is illustrated in Figure 4-3. It is clear from the figure that, to get within 1% of the reference solution, a coarse substrate discretization suffices. This result applies to the case where the geometry of the body is very simple and the force on the bottom of the cylinder and on the substrate is mostly constant. For cases like the comb-like structure in section 4.3.3 this is not the case. Another observation to be made from Figure 4-3 is that the relative error for a separation gap of 5μm did not decrease limited to 1%; this is due to the use of a finite surface to represent the *infinite* substrate. As the cylinder is brought closer to the substrate, the interaction between the cylinder and the substrate becomes more localized and the 40μm side square region used to represent the substrate becomes a better approximation of the actual substrate.

The relative error of the total vertical force on the cylinder for vertical motion as a function of the radius of the panels used to discretize the substrate plane for several separation distances between the cylinder and the substrate is illustrated in Figure 4-4. Contrary to was was observed in Figure 4-3, as the gap size is reduced, the substrate discretization had to be made much finer to achieve a given accuracy. For the smaller gaps, the finest substrate discretization did not produce a result within 1% of the reference solution. This result clearly emphasizes the advantage of using the implicit substrate representation.

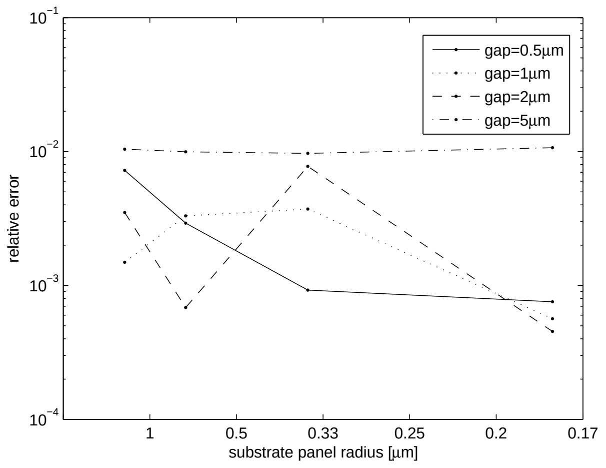


Figure 4-3: *Relative error of the drag force calculated for a cylinder moving* parallel *to the substrate. The relative error was calculated for a fixed cylinder discretization and for a set of substrate discretizations and gap sizes.*

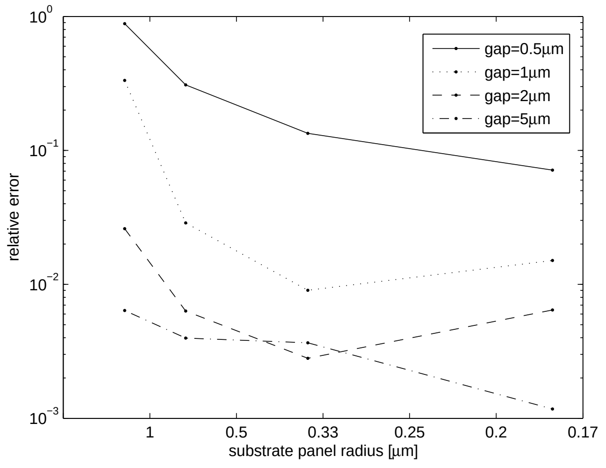


Figure 4-4: *Relative error of the drag force calculated for a cylinder moving* normal *to the substrate. The relative error was calculated for a fixed cylinder discretization and for a set of substrate discretizations and gap sizes.*

Observations

Several observations can be made from the results presented in this example. First, for lateral motion of a simple body like a cylinder, the body and the substrate discretization do not need to be made very fine to achieve reasonable accuracy. Second, for objects moving vertically, using the Stokes free space Green's function requires that the substrate discretization be made very fine to produce accurate results.

4.3.3 Substrate shadow

To compare the accuracy of the results produced by solving the discretized version of (2.14), where using the free space Green's requires the explicit discretization of the substrate, with the results of solving the discretizated version of (2.19) without an explicit substrate representation, we chose a non-smooth problem consisting of a comb-like structure with fine fingers moving over a substrate, as depicted in Figure 4-5.

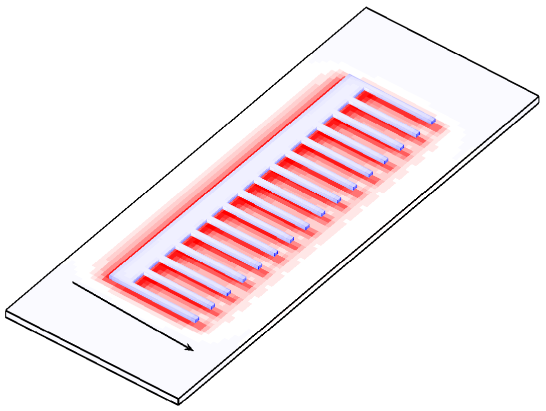


Figure 4-5: *Comb like structure moving over substrate.*

The drag force on the structure was calculated using the precorrected FFT algorithm for in-plane and out-of-plane motion for several separation distances between the comb structure and the substrate. The value of drag force on the comb structure was calculated for several combinations of discretizations for the comb and for the substrate. The separation between the comb and the substrate was also swept over a set of three values. The drag forces produced by this multidimensional sweep are summarized in Table 4.2.

Several observations can be made from the data in Table 4.2:

The variation with discretization of the results obtained for in-plane motion drag is much smaller than the variation for the values that were calculated for out-of-plane motion. For in-plane motion the results were all within 10% of the reference values, regardless of the discretization used for the substrate. Nevertheless, the error obtained using a coarse substrate discretization is larger than the error obtained using either implicit substrate discretization or a fine explicit substrate discretization.

			gap = 4μm		gap = 2μm		gap = 1μm	
comb mesh	subs. mesh	p.r. [μm]	lateral drag [pN]	vertical drag [pN]	lateral drag [pN]	vertical drag [pN]	lateral drag [pN]	vertical drag [pN]
r	n	0.32	58.65	587.84	89.44	1747.93	141.07	8251.85
c	n	2.55	57.52	554.46	87.22	1508.66	136.16	5869.81
m	n	1.27	58.21	572.23	88.60	1665.21	139.37	6976.18
f	n	0.64	58.51	582.67	89.19	1724.17	140.57	8023.81
c	c	2.69	56.94	506.57	86.12	1076.27	133.35	2366.16
m	c	1.27	57.61	532.23	87.40	1150.91	136.56	3325.24
f	c	0.64	57.92	539.91	87.92	1066.92	135.17	1119.19
c	f	0.87	57.04	554.80	86.89	1490.90	135.83	5647.43
m	f	0.87	57.73	572.34	88.27	1638.88	138.96	6652.27
f	f	0.67	58.03	582.17	88.88	1679.91	140.12	8005.46

Table 4.2: *Drag force on a comb like structure moving over a substrate. Results were obtained using different discretizations for both the structure and the substrate. Lateral drag results were obtained by setting the structure velocity to -1mm/s along the x axis. Vertical drag results were obtained by setting the structure velocity to -1mm/s along the z axis. In all cases the fluid viscosity was $\mu = 1.843 \times 10^{-5}$Pa.s. In the table* p.r. *stands for median panel radius and* c, m, *and* f *stand for coarse, medium and fine meshes.* n *stands for* no mesh *indicating that the substrate implicit solver was used.* r *stands for the reference mesh.*

Focusing on the results obtained with the implicit representation of the substrate it can be observed that, as the separation between the comb and the substrate is reduced, the drag calculated for out-of-plane motion depends strongly on the discretization resolution. This dependence indicates that, for out-of-plane motion, the solution has more variation and that a large number of constant strength panels is needed to represent the solution. Using a Galerkin test scheme slightly reduces this discretization dependence but it is still clear that a finer resolution is needed as the comb is moved closer to the substrate.

Focusing on the drag associated with out-of-plane motion calculated using explicit substrate discretizations, it can be observed that even for a relatively large separation between the comb and the substrate, the error is strongly dependent on the substrate discretization resolution. As the distance between the comb and the substrate is reduced, the accuracy of the solutions produced using a coarse substrate discretization deteriorates significantly. To reinforce the claim that the cause of this deterioration is the coarseness of the substrate discretization, consider that, for a fine substrate discretization, the results calculated using the free-space Green's function approach the results obtained using the implicit substrate representation.

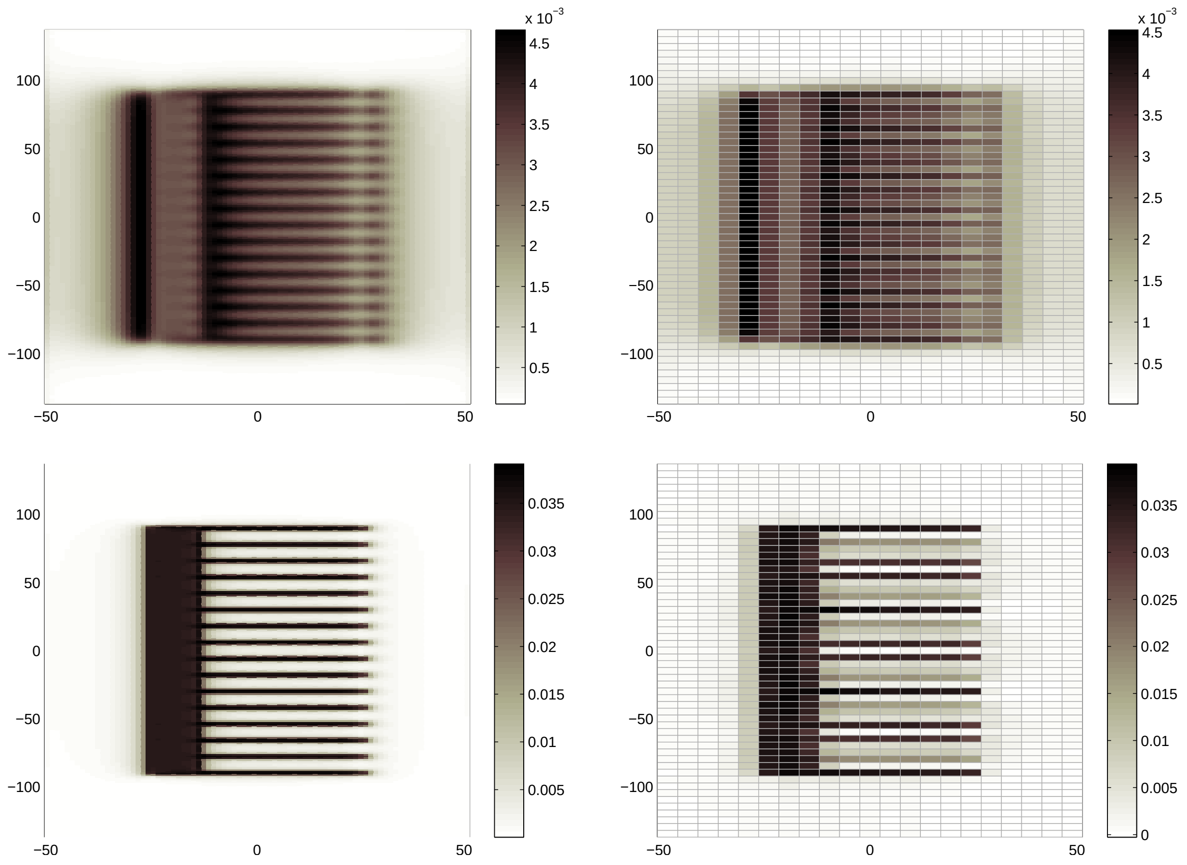


Figure 4-6: *Substrate "shadow" - Drag force density on the substrate under a comb like structure moving in-plane. The figures on the left where generated using a fine discretization for the substrate, the figures on the right where generated using a coarse discretization for the substrate. In the top two figures the comb structure was placed at 4μm above the substrate, in the bottom figures the comb structure was placed at 0.5μm above the substrate.*

To illustrate the effect of the substrate discretization resolution, Figure 4-6 shows that the forces on the substrate form a "shadow" of the comb structure above it. Comparing the shadows for the two separation distances in Figure 4-6, it is clear that the shadow becomes sharper as the comb is brought closer to the substrate. Therefore, for small separation distances, a constant coarse discretization cannot accurately represent the forces on the substrate. On the other hand, for large separation distances, where the shadow of the structure is much smoother, a coarse substrate discretization produces sufficiently accurate results.

4.3.4 MEMS accelerometer

The substrate implicit pFFT solver was used to analyze the micromachined capacitive accelerometer that was studied in [6] and [19]. Below, the drag force results reported in [6] and [19] are compared with the results calculated using our solver. For this real world example, time and memory usage are reported to reinforce observations made in section 4.3.3 and to point out some additional facts.

In [6] and [19], the combdrive was placed at a distance of 2μm above the substrate and the mesh, referred to below as mesh *c*, was used. To study the convergence of the solution with the discretization resolution mesh *c* was refined by panel subdivision to produce meshes *m*, *f* and *e*. For each mesh refinement, the combdrive was placed at a distance of 4μm, 2μm, 1μm and 0.5μm above the substrate. The drag force results and the time and memory usage for each case are summarized in Table 4.3.

Comparison to previously published results

For the case of a 2μm gap, the drag coefficient value obtained in [44], was 207.58nNm^{-1}s, corresponding to a quality factor of $Q = 29.1$. Experimentally the quality factor was measured to be $Q_{\text{exp}} = 27$. Using the substrate implicit pFFT algorithm with mesh *c*, the force estimate 217.26nNm^{-1}s was produced, after adjusting for the different value of μ used in [44], this corresponds to a $Q = 28.5$, which is marginally closer to Q_{exp}. Using a mesh with 313536 panels, labeled *e* in Table 4.3, produced a drag force of 223.6nNm^{-1}s, corresponding to $Q = 27.7$, which is consistent with the result reported in [18].

The result produced in [19] used the far more general variable order wavelet acceleration method, an implicit representation of the substrate, and the comb mesh labeled *c* in Table 4.3. In [19], the computed drag force was 214.7nNm^{-1}s. According to [19], this result was calculated in 4685s and required 4.7GB of memory. To achieve similar accuracy with an identical mesh, the precorrected FFT based solver used 281.4MB and 176.2 seconds. Although the pFFT solver outperformed the wavelet method on this fairly spatially homogeneous problem, it is well known that FFT based methods perform poorly on more

comb mesh	wall mesh	gap [μm]	p. r. [μm]	panels	memory [MB]	full time [s]	GMRES time [s]	iterations	drag [pN]
c	n	4	2.39	8418	281.4	183.9	62.1	134	-150.12
m	n	4	1.20	33672	1087.2	918.0	376.8	156	-153.07
f	n	4	0.64	78384	2507.3	1967.5	515.6	44	-154.17
e	n	4	0.32	313536	11921.8	13263.6	6432.0	44	-154.66
c	c	4	2.55	12330	401.1	534.9	355.5	270	-149.96
m	m	4	1.27	49320	1581.6	3654.4	3051.9	335	-152.87
f	f	4	1.10	140976	4461.3	4457.0	2490.1	134	-154.12
e	e	4	0.55	563904	18664.8	26399.4	16584.1	103	-154.58
c	n	2	2.39	8418	281.4	176.2	54.3	119	-217.26
m	n	2	1.20	33672	1087.2	871.7	330.3	139	-221.45
f	n	2	0.64	78384	2507.3	1976.8	527.8	45	-222.94
e	n	2	0.32	313536	11921.8	13778.9	6970.1	48	-223.60
c	c	2	2.55	12330	399.7	686.1	465.0	363	-213.62
m	m	2	1.27	49320	1509.3	4017.6	3401.4	442	-220.51
f	f	2	1.10	140976	4296.2	6499.5	4647.5	269	-222.94
e	e	2	0.55	563904	18577.5	26086.9	14928.9	129	-223.60
c	n	1	2.39	8418	281.4	168.7	46.2	103	-343.57
m	n	1	1.20	33672	1087.3	833.4	291.5	125	-351.05
f	n	1	0.64	78384	2507.3	1952.8	503.8	43	-353.54
e	n	1	0.32	313536	11921.8	13943.6	7111.7	49	-354.71
c	c	1	2.55	12330	395.7	860.5	621.9	457	-325.05
m	m	1	1.27	49320	1483.4	2207.0	1436.2	256	-347.01
f	f	1	1.10	140976	4168.3	3982.6	2164.3	139	-352.79
e	e	1	0.55	563904	18078.1	31474.9	20791.6	179	-354.33
c	n	0.5	2.39	8418	281.4	163.8	41.6	94	-577.46
m	n	0.5	1.20	33672	1087.3	821.0	278.7	120	-593.12
f	n	0.5	0.64	78384	2507.2	2110.2	663.0	57	-598.01
e	n	0.5	0.32	313536	11921.8	14487.1	7665.6	53	-600.67
c	c	0.5	2.55	12330	394.2	945.7	699.6	500	-533.30
m	m	0.5	1.27	49320	1473.0	4161.6	3305.3	500	-591.06
f	f	0.5	1.10	140976	4087.8	7199.4	5353.6	304	-596.16
e	e	0.5	0.55	563904	17835.2	29343.2	18781.4	165	-599.73

Table 4.3: *Drag force on a the movable comb of a combdrive resonator for lateral motion along the comb finger direction. The accelerometer is moving at a velocity of 1mms*$^{-1}$ *in air with* $\mu = 1.843 \times 10^{-5}$*Pa.s. Above* c *stands for coarse,* m *stands for medium,* f *stands for fine,* e *stands for finer and* n *indicates that the substrate was represented implicitly.* p.r. *stands for median panel radius. A number of 500 GMRES iterations indicates that GMRES did not converge to a relative error of* 10^{-4}.

inhomogeneous problems [15].

Observations

The results presented in Table 4.3 demonstrate that, for the same combdrive mesh resolution the pFFT algorithm using the implicit substrate discretization uses less memory and is faster than the pFFT algorithm using the free space Green's function and discretizing the substrate. This is true even though the kernels used to represent the substrate implicitly are much more complicated than the free-space Stokes Green's function. The reason for this improvement is that not only does the free-space Stokes pFFT algorithm need to account for the substrate explicitly, but also needs a larger FFT grid to encompasses the substrate and the objects above it, instead of just the objects. Furthermore, as shown in Table 4.3, the number of GMRES iterations that is required to achieve the same relative residue norm is typically much larger for the case where the substrate is represented explicitly suggesting that the system of equations associated with that formulation is poorer conditioned than the systems of equations associated with the substrate implicit method.

4.3.5 Proof mass with holes

In surface micromachined devices, parts of the device that cover large areas, such as proof masses, are often designed with several small holes that facilitate the removal of sacrificial layers and subsequent release of the device and that reduce the drag the device experiences as it moves [5].

To effectively predict the behavior of the devices, or to reduce the drag by a given desirable amount it is necessary to accurately account for the effect the holes have on the drag. For example, while designing an oscillator, to achieve a high quality factor Q one would want to maximize the reduction in the drag while minimizing the reduction in mass. While accounting for the reduction in mass due to adding holes is straightforward, accounting for the reduction in drag is more complicated since, especially for off-plane motion, the drag is strongly dependent on the size, number and distribution of the holes and on the distance between the structure and the substrate.

The effect of the holes in the drag force becomes stronger as the structures are closer to the substrate and the fluid is “forced” in the holes rather than just being pushed away from the bottom of the structure. In these cases, accurately representing the presence of the substrate is especially important. If an explicit substrate discretization method were to be used, that discretization would have to fine and hence costly. By using an implicit, accurate representation of the effects of the presence of the nearby substrate, the algorithm proposed in this chapter allows the computational resources to be better spent in refining the discretization of the actual suspended structure.

To demonstrate the use of our solver we used Comsol 3.2 to setup a set of meshes of 100μm by 100μm, 2μm thick proof mass. The maximum panel size was set to 1μm in order to guarantee that, even at a close distance to the substrate, the discretization would be fine enough to represent the solution using constant strength panels. To study the effect of the size and number of holes on the drag we used Comsol 3.2 to generate meshes with 1, 4, 9, 25 and 100 equally distributed cylindrical holes with a radius of 1μm, 2μm and 4μm. The proof masses were set at a 4μm and 1μm above the substrate. In all cases the proof mass velocity was 1mm/s moving towards the substrate. For comparison, the drag on a proof mass with no holes was also calculated. The results are summarized in Table 4.4. To further emphasize the effect of the hole radius on the force distribution on the proof mass, we present the vertical force on the bottom of a proof mass with 25 regularly space holes for holes with a radius of 1μm, 2μm and 4μm in Figure 4-7.

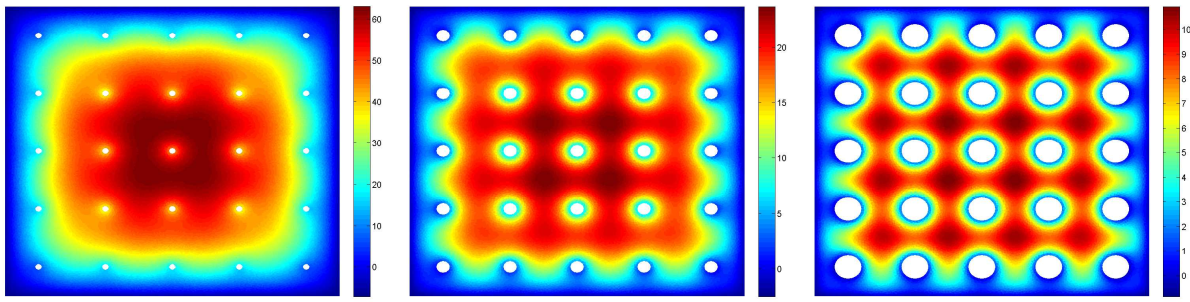


Figure 4-7: *Force on the bottom of a 100μm by 100μm by 2μm tile moving at 1μm above the substrate moving towards it at a velocity of 1mms^{-1} in a fluid with viscosity $\mu = 1.843 \times 10^{-5}$Pa.s. The tiles on the figure have 25 equally spaced holes with a radius of 1μm, 2μm and 4μm.*

num holes	h. r. [μm]	gap [μm]	p. r. [μm]	num panels	memory [MB]	full time [s]	GMRES time [s]	GMRES iterations	drag [nN]
0		4	0.56	74128	2138.5	2049.7	555.0	103	14.88
1	1	4	0.56	74314	2144.7	2070.6	579.4	103	14.83
4	1	4	0.56	74408	2148.8	2031.0	545.7	103	14.80
9	1	4	0.55	79192	2339.8	2203.0	580.6	104	14.70
25	1	4	0.55	75930	2207.6	2147.1	588.3	104	14.47
100	1	4	0.55	78454	2319.6	2524.9	817.6	137	13.39
1	2	4	0.56	74180	2140.4	2237.1	749.3	131	14.35
4	2	4	0.56	74486	2149.6	2240.4	747.9	131	13.98
9	2	4	0.56	74554	2154.1	2230.8	733.6	132	13.19
25	2	4	0.56	75406	2181.2	2241.2	721.3	132	11.07
100	2	4	0.55	77542	2250.0	2333.9	718.7	134	6.29
1	4	4	0.56	74060	2137.3	2263.2	745.2	130	12.10
4	4	4	0.56	73426	2125.0	2169.0	696.5	123	10.29
9	4	4	0.56	72494	2108.0	2176.6	710.9	129	7.56
25	4	4	0.56	69162	2040.3	2025.6	617.3	122	3.83
0		1	0.56	74128	2138.5	2048.5	569.0	107	820.67
1	1	1	0.56	74314	2144.7	2042.0	557.6	105	734.05
4	1	1	0.56	74408	2148.8	2063.1	575.4	108	685.26
9	1	1	0.55	79192	2339.8	2214.1	587.4	105	571.56
25	1	1	0.55	75930	2207.6	2077.8	532.9	100	386.42
100	1	1	0.55	78454	2319.6	2723.3	811.7	107	145.15
1	2	1	0.56	74180	2140.4	2267.1	708.0	124	618.38
4	2	1	0.56	74486	2149.6	2233.3	738.3	129	507.10
9	2	1	0.56	74554	2154.0	2265.7	761.4	134	342.78
25	2	1	0.56	75406	2181.3	2098.5	560.6	105	148.38
100	2	1	0.55	77542	2250.1	2092.3	474.6	93	30.03
1	4	1	0.56	74060	2137.4	2685.6	995.1	123	535.79
4	4	1	0.56	73426	2125.0	2236.8	755.6	133	380.58
9	4	1	0.56	72494	2108.0	2067.3	602.9	110	206.26
25	4	1	0.56	69162	2040.3	1919.5	508.3	102	59.08

Table 4.4: *Drag force on a square 100μm by 100μm tile, 2μm thick tile moving towards the substrate at a velocity of 1mms^{-1} in a fluid with viscosity $\mu = 1.843 \times 10^{-5}$kg/ms. Above* h.r. *stands for hole radius and* p.r. *stands for panel radius.*

To validate the results obtained using our solver we compared them to those produced using the Comsol 3.2 finite element code, which uses a relatively coarse volume mesh with quadratic elements. The proof mass with 4 holes with a radius of 2μm was chosen at random to perform this comparison. For the randomly chosen example, the drag calculated using Comsol 3.2 was 13.98nN for a separation distance of 4μm and 507nN for a separation distance of 1μm, which is is clearly consistent with the results reported in Table 4.4.

4.4 Conclusions and future work

A precorrected FFT accelerated algorithm for solving Stokes flow problems in the presence of a substrate was developed and demonstrated. The algorithm was validated against known theoretical, experimental and computational results and its performance was compared with previously published results.

Overall the following conclusions were drawn: the pFFT accelerated results closely match exact analytical results and results previously reported in the literature. Using the implicit substrate representation produces more accurate results with less memory and significantly less time than explicitly representing the substrate. Using the implicit substrate representation produces more accurate results because it accounts for the presence of the substrate exactly. Using the implicit substrate representation is more efficient because it requires fewer panels and because the domain that the pFFT grid must cover is much smaller than if the substrate were to be explicitly discretized. Using an implicit substrate introduces more scalar kernels, 20 instead of 6, but the speed gain obtained by eliminating the substrate and reducing the size of the pFFT grid overcomes this cost. The techniques used to extend the applicability of the pFFT algorithm to non-translation invariant kernels can be exploited in other applications.

What was a surprising and disappointing outcome of this study, which is further detailed in Chapter 5 is that out-of-plane motion excites equation modes that reveal the need to refine the structure discretization as the distance to the substrate decreases. Simulation of out-of-plane motion also revealed that, when using an explicit substrate, the substrate discretization must be refined faster than than structure discretization for results to match the results obtained using implicit substrate discretization. So implicit substrate representation has benefits but does not entirely decouple structure discretization from distance to the substrate.

As future work we propose supporting higher order panel force distributions to reduce the number of panels and improve convergence.

Chapter 5

A surprising result

The Stokes substrate Green's function can be used to implicitly represent the substrate. By using the Stokes substrate Green's function, a boundary integral formulation involving only the structures above the substrate can be produced. The boundary integral equations are discretized by approximating the geometry of the problem by a set of flat panels; by limiting the solution space to only constant force distributions on each panel and by testing the equations only at the centroid of each panel.

By removing the need to explicitly represent the substrate it was expected that, unless the solution became more complicated, the panel discretization for the structures above the substrate would not need to be refined as the distance between the structures and the substrate decreases. In this chapter, using a very simple example, we demonstrate that vertical motion activates modes of the equation that require the discretization to be refined, regardless of the smoothness of the solution.

In the following, the example of a cylinder with a radius of 10μm over a substrate is used to study the effect of discretization refinement on the accuracy of the calculated drag force for both horizontal and vertical motion.

To study the effect of the cylinder discretization and distance to the substrate on the accuracy of the calculated drag force a set of cylinder discretizations were constructed using Comsol 3.2. Each cylinder mesh was then placed at a distance of 5μm, 2μm, 1μm and 0.5μm from the substrate and the drag force was calculated. The solutions calculated for the finer cylinder discretizations, with a median panel radius of 0.14μm, were used as reference. For the gap size of 0.5μm, a finer discretization of the cylinder, with a median panel radius of 0.07μm, was used to calculate the reference drag forces.

5.1 Lateral motion

For a cylinder moving in a direction parallel to the substrate, the drag force is largest on the bottom of the cylinder. For a small enough gap, the drag on the bottom of the cylinder can be accurately predicted using the Couette flow model and is given by

$$F = V\mu a^2/h,$$

where V is the velocity, μ is the fluid viscosity, a is the cylinder radius and h is the gap between the cylinder and the substrate.

The error of the computed drag force on the bottom of the cylinder, as a function of the median radius of panels used to represent the cylinder is illustrated in Figure 5-1. For these computations, the Green's function in (2.16) is used to implicitly represent the substrate. Note that as the discretization is refined, the relative error is reduced at roughly the same rate, regardless of the gap between the cylinder and the substrate.

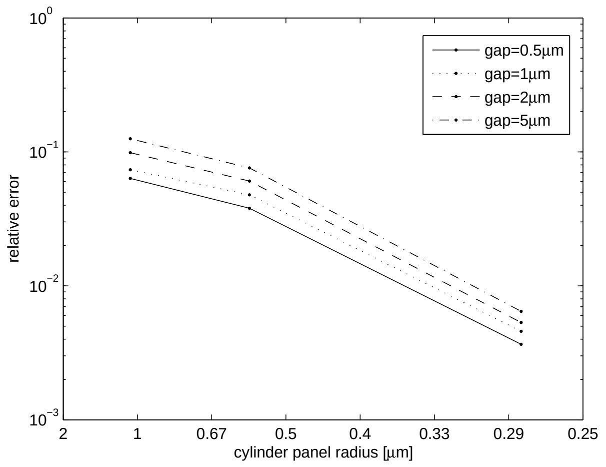


Figure 5-1: *Relative error of the drag force calculated for a cylinder moving parallel to a substrate for several cylinder discretizations for several gap sizes. The substrate was represented implicitly.*

5.2 Vertical motion

For a cylinder near the substrate that is moving with a velocity normal to the substrate, the dominant force is the pressure on the bottom of the cylinder. For a small gap h between the cylinder and the substrate, the pressure on the bottom of the cylinder of radius a moving at a velocity V in the direction normal to the substrate can be approximated as

$$P(r) = 3\mu V h^{-3}(\eta^2 - a^2) \tag{5.1}$$

i.e. a quadratic function of η, the radial distance to the cylinder's axis, that is scaled by a factor that is inversely proportional to the cube of the gap [2]. Through our numerical experiments we have observed that (5.1) is a very good approximation to the pressure on the bottom of a cylinder, except very near the cylinder edges. Moreover, (5.1) becomes more accurate as the gap between the cylinder and the substrate shrinks. Therefore, since the solution is approximately a quadratic scaled by a size dependent factor, it is expected that a constant force panel discretization should represent the solution to the same relative accuracy independent of the gap size. Surprisingly, this is not the case. To accurately match the reference solution, the cylinder discretization must be made finer as the gap shrinks. Such a result is surprising because the need to refine the discretization with the reduction of the gap size is *not* driven by the need to more accurately represent the solution. This can be observed in Figure 5-2, where the error of the vertical force on the cylinder is plotted as a function of the median radius of the panels. Contrary to the result in Figure 5-1, the relative error for the smaller gaps is much larger than the relative error for the larger gaps. For the smaller gap of 0.5μm a very fine discretization was necessary to accurately match the reference solution. Moreover, for a given discretization, as the gap size is reduced, the number of GMRES iterations required to achieve a given convergence tolerance increases. This increase in the number of iterations suggests that the discretized system's condition number is rising.

Since the need to make the discretization finer is not driven by the need to more accurately represent the solution, one possible explanation for the need to increase the number of pan-

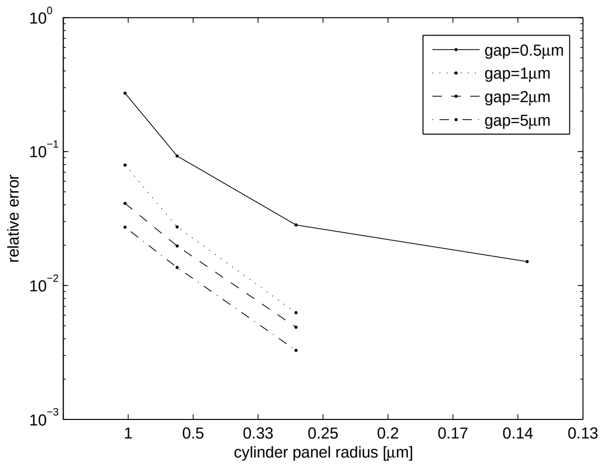


Figure 5-2: *Relative error of the drag force calculated for a cylinder moving parallel to a substrate for several cylinder discretizations for several gap sizes. The substrate was represented implicitly.*

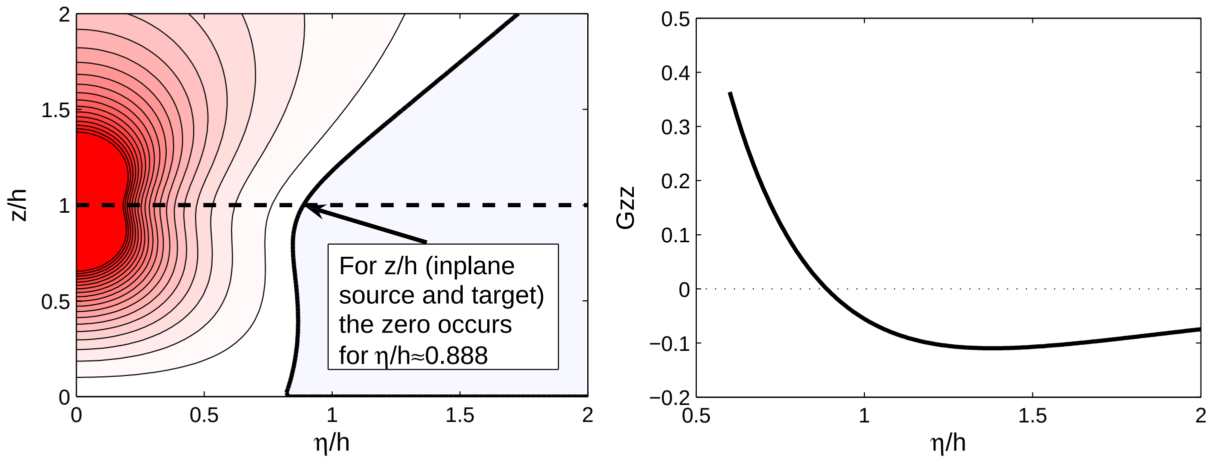


Figure 5-3: *Velocity along the z direction due to a force along the z direction located at $(0, 0, h)$ evaluated a point $(\eta\cos\theta, \eta\sin\theta, z)$, where η is the radial distance from the source point to the evaluation point. The figure on the left illustrates the vertical velocity field; the darker line marks the points where the vertical velocity changes sign. The figure on the right illustrates the vertical velocity as a function of radial position for evaluation points in the same plane as the source.*

els as the gap size is reduced is the "behavior" of the Green's function. A possible source for the behavior is the way that the velocity due to a constant vertical force distribution on a flat panel changes as the separation distance between the panel and the substrate is reduced. The vertical velocity due to a vertical force applied at a distance h above a substrate behaves as illustrated in Figure 5-3, the velocity is positive near the point where the force is applied but becomes negative at a radial distance of about $0.888h$. For a constant strength panel of a given size, as h is reduced and becomes smaller than the panel size, the velocity due to a force on one part of the panel will cancel out the velocity due to the force on other parts of the panel. This effect is clearly illustrated in Figure 5-4 where, for the larger panel size and small h, the velocity field due to a constant strength panel is greatly reduced and exhibits a very sharp and complicated behavior. On the other hand, the velocity fields produced by smaller panels, not subject to self-cancellation at the separation distances in the figure, keep the smooth shape. The effect is further illustrated in Figure 5-5 where, for a gap of $h = 0.25\mu$m, the vertical velocity field due to each panel is plotted in the same scale for easier comparison.

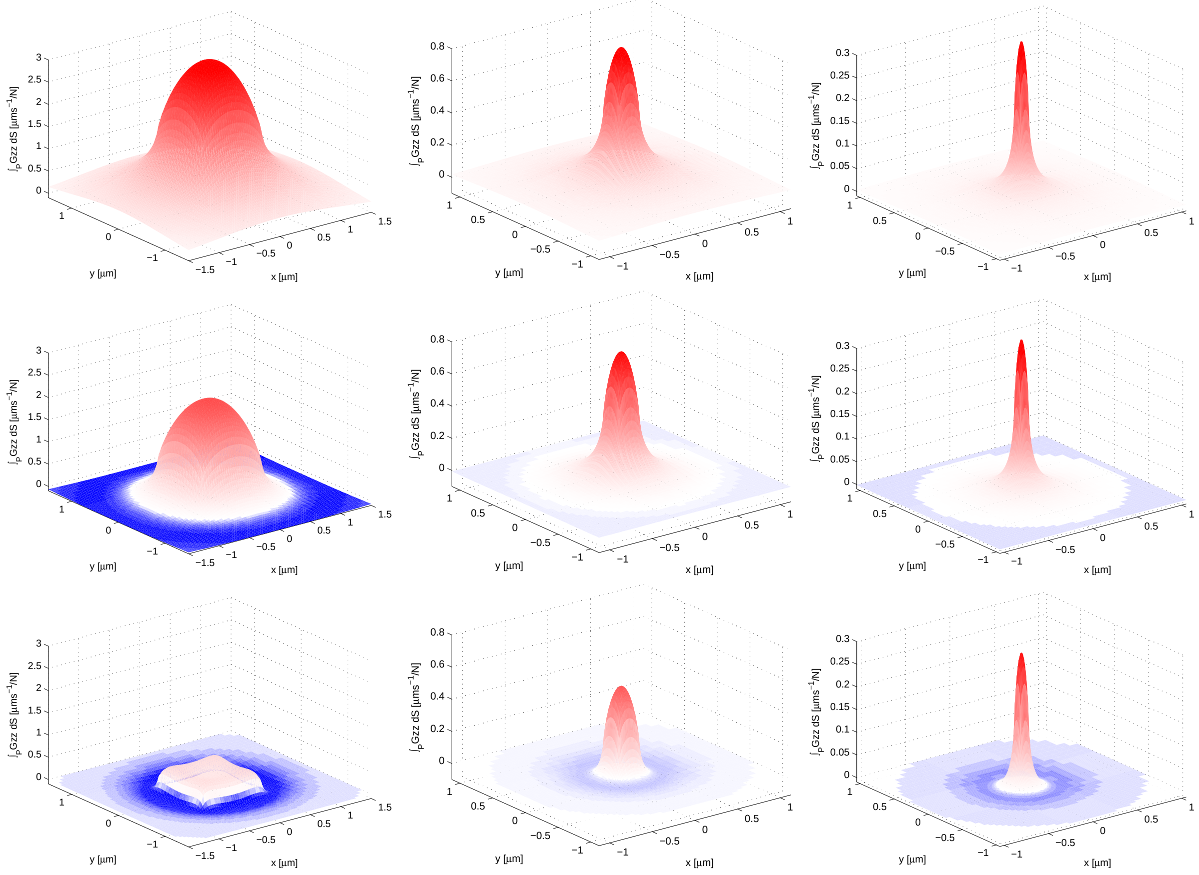
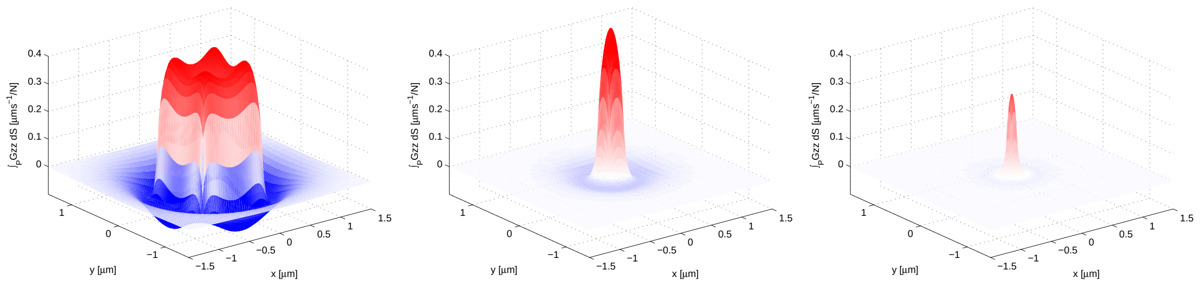


Figure 5-4: *This figure illustrates the self-cancellation that occurs for larger panel sizes at smaller gaps by depicting the vertical velocity generated by a constant force panel for three square panels of varying sizes placed at three gap distances from the substrate. The results on the first row correspond to a gap size of 4μm, on the second row the gap is 1μm and on the third row the gap is 0.25μm. The results on the left column correspond to a panel side of 1μm; on the middle column to a panel side of 0.25μm and on the right column to a panel side of 0.1μm.*



panel side = 1μm panel side = 0.25μm panel side = 0.1μm

Figure 5-5: *This figure illustrates the self-cancellation that occurs for the vertical velocity due to a uniform vertical force on panels above a substrate. The panels on the three figures where placed at a distance of 0.25μm from the substrate.*

5.3 Observations

Several observations can be made from the results presented in the cylinder example. First, for lateral motion of a simple body like the cylinder, coarse object discretization achieves a reasonable accuracy. Second, for objects moving vertically, accurate results are produced only if the discretization is refined as the gap is reduced, *even though the force distribution smoothness is unchanged*. This behavior is observed regardless of whether the substrate is represented implicitly or explicitly. The practical impact of this observation is that representing the substrate implicitly has many benefits but unfortunately does *not* completely decouple discretization fineness from distance to the substrate.

Chapter 6

Time domain simulation

The simulation of objects moving in Stokes or creeping flow is a convenient tool for the design of microfluidic devices such as cell traps [21, 22, 23, 24] and micromixers [25]. The time domain simulation MEMS devices is also very important for the design of devices such as micromirrors [26].

As was reviewed in Chapter 2, for problems where the length scale, L, the characteristic velocity, V, the fluid viscosity μ and the fluid density ρ are such that the Reynolds number Re=$LV\rho/\mu$ is much smaller than one, the inertial terms of the Navier Stokes equations can be neglected and the Stokes equations can be used.

The Stokes equations state that the pressure, viscous forces and body forces are at balance regardless of the history of flow, even though the boundaries of the flow maybe changing in time [1]. The Reynolds number, Re, is the ratio between the time constant for the diffusion of momentum in the fluid $\tau_D = \rho L^2/\mu$ and the time constant for convection $\tau_C = L/V$. When the Reynolds number is small, and there are no abrupt changes in the fluid velocity, momentum diffuses throughout the fluid domain much faster than the configuration of the flow is changing due to the evolution of its boundaries [1]. Therefore, in these conditions, a quasi-static approach for analyzing the time evolution of the system is appropriate [25].

Several methods exist for the calculation of drag forces on objects immersed in Stokes flow: finite differences [8], immersed boundary methods [9], the finite element method [10] and the boundary element method [1]. Since, for Stokes flow, the fluid structure only depends on the boundary configuration at the time point of interest, the boundary element method is a particularly suitable approach. Moreover, for problems where one is interested in the time domain evolution of a system, the boundary element method has the advantage that remeshing the domain at each step is not necessary. Furthermore, using the boundary element method with appropriate Green's functions it is often possible to drive the motion of the objects in the flow by specifying a background flow without having to explicitly discretize the surface of the microfluidic channel or other boundaries that, in other methods, would just be used to drive the bulk fluid.

It would therefore seem that, to simulate the motion of objects in Stokes flow one would simply need to use a boundary element solver to calculate the drag force on each object in the fluid and to use these forces to update the velocity and position of the objects by integrating the equations of motion. However, for small length scales, such as those present in MEMS and microfluidic devices, the ratio of the drag force and the mass of the bodies is such that the time constant associated with transferring momentum between an object and the surrounding fluid is very small. For typical geometries, the time scale for momentum transfer between the objects and the fluid is much smaller than the timescale at which the objects move through the devices, which is usually the time scale of interest in simulation. The existence of this very small time scale makes the problem stiff and severely limits the step sizes that explicit time integration schemes can use. Constrained to using very small time steps, even though the actual solution of interest is smooth, the simulation of realistic problems becomes too expensive, even if efficient accelerated boundary element methods are used.

Typically, stiffness is dealt with by using implicit time integration methods. However, since the forces on the surface of the objects depend on the position, orientation and velocity of the objects, using an implicit time integration method would require solving a possibly large boundary element problem involving a non-linear dependence of the forces of the object on the object position. In this chapter, we demonstrate that the small time constants that limit the time steps that can be used by explicit time integration methods are due to the relation between the velocity of the objects and the drag force on their surfaces and not to the rate at which the fluid structure changes as the objects are convected through the fluid.

To deal with stiffness without incurring in the excessive cost of solving a non-linear equation for the forces on the surface of the object at each time step, we introduce a method to couple a time-stepping scheme that updates the velocity implicitly and the position explicitly with a boundary element solver for Stokes flow. The velocity implicit time stepping scheme enables the simulation of the motion of objects using large time steps. We demonstrate the stability of our method and apply it to a set of microfluidic applications. To deal with problems involving collisions, contacts and friction we coupled our velocity-implicit time integration method with the freely available rigid body physics library ODE [27].

This chapter is structured as follows: first, in Section 6.1, the boundary integral equation formulation of the Stokes flow problems presented in Chapter 2 is extended to support the definition of background flows and problems involving structures protruding above a substrate or holes on a substrate; in Section 6.2 and Section 6.3 the issue of stiffness in Stokes flow is illustrated for the case of a sphere in infinite flow; in Section 6.4, the time-integration schemes presented in Section 6.3 are coupled with rigid body mechanics and a boundary element solver for the Stokes drag force; in Section 6.5 details about the algorithm and of how collisions and friction are dealt with are presented; in Section 6.6 optimizations aiming to reuse part of the pFFT solver structures from step to step are presented; finally in Sections 6.7 and 6.8 results presented and discussed, conclusions are drawn and directions for future work are proposed.

6.1 Boundary integral formulation

This section is an extension of the presentation in Section 2.1. In this section we describe how to formulate boundary integral equations for calculating the drag force on objects in Stokes flow in the presence of a background flow for three special cases that are of special interest for the simulation of objects moving in microfluidic systems. First, in Section 6.1.1 a boundary integral equation for the drag force on an object immersed in a background flow is described. In Section 6.1.2 a formulation for calculating the drag on objects immersed in

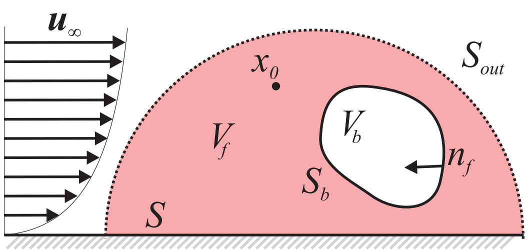


Figure 6-1: *Stokes flow around a rigid body. In the figure* $\mathbf{u}^\infty$ *represents the flow in the absence of* V_b*.* V_f *is the fluid volume,* V_b *is the volume of the rigid body, Sb is the boundary between the fluid and the rigid body,* S *is the substrate and* S_{out} *represents a surface in the fluid that is considered to be arbitrarily distant from the other features. The source point* $\mathbf{x}_0$ *is represented inside the fluid.*

a background flow over over a substrate with protuberant structures is presented; Finally, in Section 6.1.3 we introduce a formulation for the drag force on objects immersed in a background flow near a hole in a substrate.

6.1.1 Background flow

A natural way to drive the motion of an object in a flow it is to introduce a background flow. The background flow is the solution of the Stokes flow problem in the absence of the perturbation introduced by the presence of the object. By using the linearity of the Stokes equations a boundary integral equation for the drag on an object moving in a background flow can be obtained combining a boundary integral equations for the background flow itself and a boundary integral equation for the perturbation flow [1].

To derive the Stokes flow boundary integral equations for the motion of a rigid body V_b in the presence of a background flow $\mathbf{u}^\infty$ as illustrated in Figure 6-1. We will first assume that the source point, $\mathbf{x}_0$, for the Stokes Green's function is inside the fluid volume V_f as illustrated in Figure 6-1 , we will then consider the case where $\mathbf{x}_0$ is on S_b, the boundary of the object.

First we integrate the Lorentz reciprocity identity for the perturbation flow identity over V_f excluding an infinitesimal sphere around the source point $\mathbf{x}_0$ in V_f and get the boundary integral equation

$$\int_{S_b} \mathbf{G}(\mathbf{x}, \mathbf{x}_0)\mathbf{f}^D dS(\mathbf{x}) - \mu \int_{S_b} \mathbf{T}(\mathbf{x}, \mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^D(\mathbf{x})dS(\mathbf{x}) = -8\pi\mu\mathbf{u}^D(\mathbf{x}_0). \quad (6.1)$$

Due to the no slip boundary condition, the velocity on S_b is $\mathbf{u}(\mathbf{x}) = \mathbf{u}^\infty + \mathbf{u}^D = \mathbf{u}_b + \boldsymbol{\omega}_b \times \mathbf{x}$, where $\mathbf{u}_b$ and $\boldsymbol{\omega}_b$ are the linear and angular velocity of object b. Since $\mathbf{u}$ is a rigid body motion and we are excluding an infinitesimal spherical region of fluid around $\mathbf{x}_0$, the double layer integral in (6.1) becomes

$$\int_{S_b} \mathbf{T}(\mathbf{x}, \mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^D(\mathbf{x})dS(\mathbf{x}) = -\int_{S_b} \mathbf{T}(\mathbf{x}, \mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^\infty(\mathbf{x})dS(\mathbf{x})$$

yielding

$$\int_{S_b} \mathbf{G}(\mathbf{x}, \mathbf{x}_0)\mathbf{f}^D(\mathbf{x})dS(\mathbf{x}) - \mu \int_{S_b} \mathbf{T}(\mathbf{x}, \mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^\infty(\mathbf{x})dS(\mathbf{x}) = -8\pi\mu\mathbf{u}^D(\mathbf{x}_0).$$

By integrating the Lorentz reciprocity identity for the background flow on the interior of the body, V_b, we get

$$\int_{S_b} \mathbf{G}(\mathbf{x}, \mathbf{x}_0)\mathbf{f}^\infty(\mathbf{x})dS(\mathbf{x}) - \mu \int_{S_b} \mathbf{T}(\mathbf{x}, \mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^\infty(\mathbf{x})dS(\mathbf{x}) = 0$$

where there is no free term in the velocity because the source point is in V_f and we are integrating over V_b. The negative sign preceding the normal is meant to emphasize the that the integral outer normal is $-\mathbf{n}_f$.

Adding the two equations yields

$$\int_{S_b} \mathbf{G}(\mathbf{x}, \mathbf{x}_0)(\mathbf{f}^\infty + \mathbf{f}^D)(\mathbf{x})dS(\mathbf{x}) = -8\pi\mu\mathbf{u}^D(\mathbf{x}_0) = -8\pi\mu(\mathbf{u}_b + \boldsymbol{\omega}_c \times \mathbf{x}_0 - \mathbf{u}^\infty(\mathbf{x}_0))$$

that can simply be written as

$$\int_{S_b} \mathbf{G}(\mathbf{x}, \mathbf{x}_0)\mathbf{f}(\mathbf{x})S(\mathbf{x}) = -8\pi\mu(\mathbf{u}_b + \boldsymbol{\omega}_c \times \mathbf{x}_0 - \mathbf{u}^\infty(\mathbf{x}_0)) \tag{6.2}$$

This equation is also valid in the more interesting case when $\mathbf{x}_0$ is on S_b . To prove this consider the two integration regions illustrated in Figure 6-2.

Integrating the Lorentz reciprocity identity over V_f with $\mathbf{x}_0$ on S_b for the perturbation flow

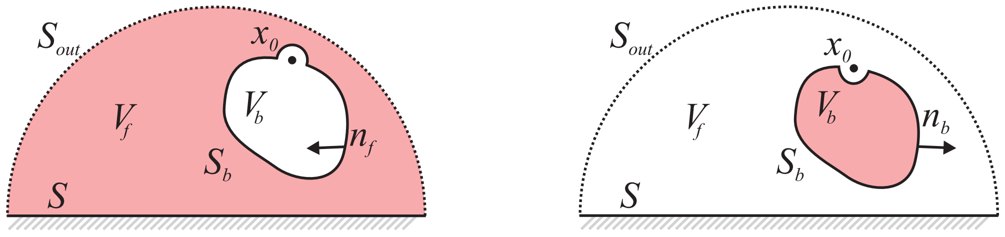


Figure 6-2: *Integration region with source point* $\mathbf{x}_0$ *on the boundary* S_b*. The exclusion region is an infinitesimal hemisphere.*

yields

$$\int_{S_b} \mathbf{G}(\mathbf{x},\mathbf{x}_0)\mathbf{f}^D d(\mathbf{x})S(\mathbf{x}) - \mu \int_{S_b}^{\mathrm{PV}} \mathbf{T}(\mathbf{x},\mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^D(\mathbf{x})dS(\mathbf{x}) = -4\pi\mu\mathbf{u}^D(\mathbf{x}_0).$$

Since the source point is on the boundary and the velocity is rigid body we have

$$\int_{S_b}^{\mathrm{PV}} \mathbf{T}(\mathbf{x},\mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))(\mathbf{u}-\mathbf{u}^\infty)(\mathbf{x})dS(\mathbf{x}) = -\int_{S_b}^{\mathrm{PV}} \mathbf{T}(\mathbf{x},\mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^\infty(\mathbf{x})dS(\mathbf{x}) - 4\pi\mu\mathbf{u}(\mathbf{x}_0),$$

which, for $\mathbf{u}(\mathbf{x}) = \mathbf{u}_b + \boldsymbol{\omega}_b \times \mathbf{x}$, yields

$$\begin{aligned}\int_{S_b} \mathbf{G}(\mathbf{x},\mathbf{x}_0)\mathbf{f}^D(\mathbf{x})dS(\mathbf{x}) + \mu \int_{S_b}^{\mathrm{PV}} \mathbf{T}(\mathbf{x},\mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^\infty(\mathbf{x})dS(\mathbf{x}) = \\ -4\pi\mu\mathbf{u}^D(\mathbf{x}_0) - 4\pi\mu(\mathbf{u}_b + \boldsymbol{\omega}_c \times \mathbf{x}_0)\end{aligned} \tag{6.3}$$

Integrating the Lorentz reciprocity identity for the background flow over V_b, when the source point is on S_b, yields

$$\int_{S_b} \mathbf{G}(\mathbf{x},\mathbf{x}_0)\mathbf{f}^\infty(\mathbf{x})dS(\mathbf{x}) - \mu \int_{S_b}^{\mathrm{PV}} \mathbf{T}(\mathbf{x},\mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^\infty(\mathbf{x})dS(\mathbf{x}) = 4\pi\mu\mathbf{u}^\infty(\mathbf{x}_0) \tag{6.4}$$

Adding (6.3) and (6.4) produces (6.2), that only differs from the boundary integral equation without a background flow because of the $\mathbf{u}^\infty$ term on the right hand side.

6.1.2 Protuberances on substrate

For the case of microfluidic devices such as the pachinko cell traps [21][22][23], the moving objects S_c and the fixed structures S_p are close to the bottom of a microfluidic channel but far enough from the other device walls that the flow field near these walls is not significantly affected by the presence of S_c and S_p. In these conditions, the flow field near S_c and S_p can be represented by the sum of a background flow $\mathbf{u}^\infty$, the Stokes flow solution in the absence of S_c and S_p, and a perturbation flow $\mathbf{u}^D$ that, added to $\mathbf{u}^\infty$ satisfies the no-slip velocity boundary conditions on the surface of S_c and S_p. Since S_c and S_p are near the bottom of the microfluidic device, S, the problem geometry is locally similar to the semi-infinite problem illustrated in Figure 6-3. When dealing with structures over a substrate S, the Stokes substrate Green's function [1, 45] can be used to represent the substrate implicitly. By using the Stokes substrate Green's function only the structures above the substrate need to be discretized greatly reducing the memory and time required for calculating the drag force on the objects moving in the flow [45].

To formulate a set of boundary integral equations for the forces on S_c and S_p we will first consider only S_p and then merge the resulting equation with (6.2). We will first consider the case where $\mathbf{x}_0$ is in the fluid volume V_f and afterwards will consider the case where $\mathbf{x}_0$ is on S_p.

Integrating the Lorentz reciprocity identity for the perturbation flow over V_f excluding a spherical region around $\mathbf{x}_0$ we get

$$\int_{S_b \bigcup S} \mathbf{G}(\mathbf{x},\mathbf{x}_0)\mathbf{f}^D dS(\mathbf{x}) - \mu \int_{S_b \bigcup S} \mathbf{T}(\mathbf{x},\mathbf{x}_0)(-\mathbf{n}_f(\mathbf{x}))\mathbf{u}^D(\mathbf{x}) dS(\mathbf{x}) = -8\pi\mu\mathbf{u}^D(\mathbf{x}_0).$$

Since $\mathbf{u}^D = -\mathbf{u}^\infty$ on S_p and $\mathbf{u}^D = \mathbf{u}^\infty = \mathbf{0}$ on S, the equation can be written as

$$\int_{S_b \bigcup S} \mathbf{G}(\mathbf{x},\mathbf{x}_0)\mathbf{f}^D dS(\mathbf{x}) - \mu \int_{S_b \bigcup S} \mathbf{T}(\mathbf{x},\mathbf{x}_0)\mathbf{n}_f(\mathbf{x})\mathbf{u}^\infty(\mathbf{x}) dS(\mathbf{x}) = -8\pi\mu\mathbf{u}^D(\mathbf{x}_0). \tag{6.5}$$

Integrating the Lorentz reciprocity identity for the background over V_p yields

$$\begin{aligned}&\int_S \mathbf{G}(\mathbf{x},\mathbf{x}_0)\mathbf{f}^D(\mathbf{x}) dS(\mathbf{x}) + \int_{S_b} \mathbf{G}(\mathbf{x},\mathbf{x}_0)\mathbf{f}^\infty(\mathbf{x}) dS(\mathbf{x}) \\ &+ \int_{S_p} \mathbf{G}(\mathbf{x},\mathbf{x}_0)(\mathbf{f}^\infty + \mathbf{f}^D)(\mathbf{x}) dS(\mathbf{x}) = -8\pi\mu\mathbf{u}^D(\mathbf{x}_0)\end{aligned}$$

If $\mathbf{G}$ is the Stokes substrate Green's function the first two integrals are zero and we get

$$\int_{S_p} \mathbf{G}(\mathbf{x},\mathbf{x}_0)\mathbf{f}(\mathbf{x})dS(\mathbf{x}) = -8\pi\mu(\mathbf{u}(\mathbf{x}_0) - \mathbf{u}^{\infty}(\mathbf{x}_0)), \tag{6.6}$$

which is equivalent to (6.2). The same identity holds when $\mathbf{x}_0$ is on S_p (the proof is similar to the corresponding proof in the previous section).

A boundary integral equation for the force on S_c and S_p is given by

$$\int_{S_p \bigcup S_c} \mathbf{G}(\mathbf{x},\mathbf{x}_0)\mathbf{f}(\mathbf{x})dS(\mathbf{x}) = \begin{cases} -8\pi\mu(\mathbf{u}_b + \boldsymbol{\omega}_c \times \mathbf{x}_0 - \mathbf{u}^{\infty}(\mathbf{x}_0)) & \text{for } \mathbf{x}_0 \text{ on } S_c \\ 8\pi\mu\mathbf{u}^{\infty}(\mathbf{x}_0) & \text{for } \mathbf{x}_0 \text{ on } S_p \end{cases} \tag{6.7}$$

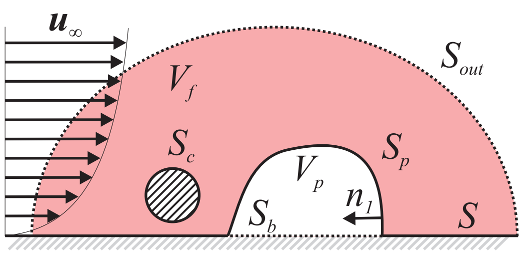


Figure 6-3: *Stokes flow around a protuberance on the substrate.* $\mathbf{u}^{\infty}$ *represents the flow in the absence of the protuberance.* V_f *is the fluid volume,* V_p *is the volume of the protuberance,* S_p *is the boundary between the fluid and the protuberance,* S *is the substrate, except for the part that is covered by the protuberance,* S_b *is the part of the substrate that is covered by the protuberance,* S_c *is the surface of a rigid body moving in the flow and* S_{out} *represents a surface in the fluid that is considered to be arbitrarily distant from the other features. The source point is not represented in this figure.*

6.1.3 Holes in the substrate

Other microfluidic trap geometries, such as the microwell cell traps presented in [24], can be described as ”holes” in the bottom of a microfluidic channel as illustrated in Figure 6-4. Since discretizing a significant portion of the substrate near the hole would be expensive and there would be no way to impose a ”natural” background flow because the structures of interest are in a region where the background flow (e.g. fully developed channel flow) is not defined, it became necessary to explore alternative formulations.

As in the case for protuberances over a substrate, a formulation using the Stokes substrate

Green's function can be used to eliminate the need to explicitly discretize the substrate. Such a formulation can be obtained by separating the problem into an interior region V_i and an exterior region V_f as illustrated in Figure 6-4. Applying the Lorentz reciprocity

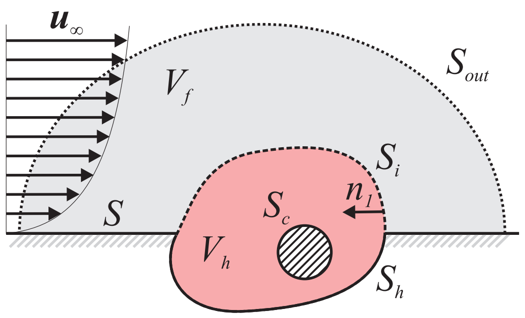


Figure 6-4: *Integration regions for the formulation of boundary integral equations for a problem involving traps formed by a hole on a substrate and objects moving in a background flow.*

identity on the exterior region V_f, yields a boundary integral equation on S_i for the force on the interface an the fluid velocity. A background flow can be introduced in the exterior region, where it is a valid solution of the Stokes flow problem. The equations for the exterior region are coupled to the equations for the interior region by equating the velocity and force on both sides of S_i. The boundary integral equations for the interior region V_i use the free-space Stokes Green's function and do not consider the background flow explicitly. The resulting system of boundary integral equations is

$$-\mu\int_{S_i}\mathbf{T}(\mathbf{x},\mathbf{x}_0)\mathbf{u}^\infty(\mathbf{x})dS+\int_{S_{\text{i+h+c}}}\mathbf{G}(\mathbf{x}_0,\mathbf{x})\mathbf{f}(\mathbf{x})dS=\begin{cases}-4\pi\mu\mathbf{u}^\infty(\mathbf{x}_0) & \text{for } \mathbf{x}_0 \text{ on } S_i\\ -8\pi\mu\mathbf{u}_c(\mathbf{x}_0) & \text{for } \mathbf{x}_0 \text{ on } S_c\\ 0 & \text{for } \mathbf{x}_0 \text{ on } S_h\end{cases}$$

$$-\mu\int_{S_i}\mathbf{T}^W(\mathbf{x},\mathbf{x}_0)\mathbf{u}(\mathbf{x})dS+\int_{S_\text{i}}\mathbf{G}^W(\mathbf{x}_0,\mathbf{x})\mathbf{f}(\mathbf{x})dS=-4\pi\mu(\mathbf{u}(\mathbf{x}_0)-\mathbf{u}^\infty(\mathbf{x}_0)) \quad \text{for } \mathbf{x}_0 \text{ on } S_i \tag{6.8}$$

Even though (6.8) accurately accounts for the background flow and the presence of the substrate, this formulation requires introducing an arbitrary boundary in the fluid and uses the Stokes substrate double layer Green's function. Implementing a solver (6.8) would require a significant implementation and computational effort because the Stokes substrate double layer Green's function has many, relatively complicated scalar entries. Accelerating

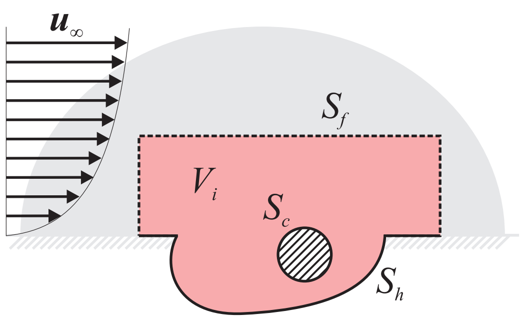


Figure 6-5: *Integration regions for formulation of problem with hole on a substrate using the free space Stokes flow Green's function. The surface S_i is expected to be far enough from S_h and S_c that the velocity at S_i is very close to the background velocity.*

the matrix vector products involving the Stokes substrate double layer Green's function using the pFFT algorithm can use exactly the same modified projection and interpolation schemes described in Chapter 4 but, in the absence of some automation or code generation step, manually accounting for the large number of of kernels that would result from such the decomposition scheme used for the single layer Green's function would prove to be quite cumbersome and error prone.

Alternatively, an approximate solution can be obtained by imposing velocity boundary conditions on S_i stating that the velocity on S_i is the background flow velocity $\mathbf{u}^\infty$. This approximation is valid if S_i is far enough from S_h and S_c that the velocity at S_i is very close to the background velocity. Due to the short development length for Stokes flow S_i does not need to be at a *very* large distance from S_h and S_c for this approximation to be valid. The integration region for this approximate formulation is illustrated in Figure 6-5. Integrating the Lorentz reciprocity relation over V_i yields

$$\mu \int_{S_i} \mathbf{T}(\mathbf{x},\mathbf{x}_0)\mathbf{u}^\infty(\mathbf{x})dS + \int_{S_{\mathrm{i+h+c}}} \mathbf{G}(\mathbf{x}_0,\mathbf{x})\mathbf{f}(\mathbf{x})dS = \begin{cases} -4\mu\pi\mathbf{u}^\infty(\mathbf{x}_0) & \text{for } \mathbf{x}_0 \text{ on } S_i \\ -8\mu\pi\mathbf{u}_c(\mathbf{x}_0) & \text{for } \mathbf{x}_0 \text{ on } S_c \\ 0 & \text{for } \mathbf{x}_0 \text{ on } S_h \end{cases} \tag{6.9}$$

where $\mathbf{u}_c$ represents the rigid body velocity on the surface S_c. The integration region for this simplified problem is limited to V_i and no exterior region is considered.

If the integral of the background flow u^∞ over S_i is zero, which is true as long as $\mathbf{u}^\infty$ is an incompressible flow, the double layer integral over S_i can be eliminated by integrating the

Lorentz reciprocity identity over V_f and V_s and subtracting the resulting boundary integral equations from (6.9) results in

$$\int_{S_i} \mathbf{G}(\mathbf{x}_0,\mathbf{x}) \underbrace{\left(\mathbf{f}(\mathbf{x}) - \mathbf{f}_i^{\text{out}}(\mathbf{x})\right)}_{\hat{\mathbf{f}}_i(\mathbf{x})} dS + \int_{S_{\text{h+c}}} \mathbf{G}(\mathbf{x}_0,\mathbf{x})\mathbf{f}(\mathbf{x}) dS = \begin{cases} -8\pi\mu\mathbf{u}^{\infty}(\mathbf{x}_0) & \text{for } \mathbf{x}_0 \text{ on } S_i \\ -8\pi\mu\mathbf{u}_c(\mathbf{x}_0) & \text{for } \mathbf{x}_0 \text{ on } S_c \\ 0 & \text{for } \mathbf{x}_0 \text{ on } S_h \end{cases} \tag{6.10}$$

that can be solved for $\hat{\mathbf{f}}$ on S_i and for $\mathbf{f}$ on S_h and S_c. Note that, because of the method used to eliminate the double layer kernel, the force $\hat{\mathbf{f}}$ on S_i is not the the same force $\mathbf{f}$ that is obtained by solving (6.9). Fortunately, the forces on S_c and S_h, which are the forces of interest for calculating the time evolution of the system, are not affected.

6.1.4 Boundary element method

Except for very simple geometries and boundary conditions, there are no analytical solutions for the boundary integral equations (6.2), or (6.7), or (6.10). These boundary integral equations can be solved approximately by discretizing the surfaces, constraining the solution to be lie in a finite dimensional vector space, and enforcing the satisfaction of the equations at a finite number of points, collocation, or enforcing that the integral of the residue, multiplied by test functions, be zero over the discrete elements, Galerkin. Having discretized the surface of the problem into n_p elements, a straightforward implementation of the boundary element method would require the calculation of the interactions between the n_p panels requiring $O(n_p^2)$ storage, which becomes prohibitively expensive in both computation time and memory when applied to large engineering problems. Accelerated boundary element methods using the multipole method [16][17][20], the multiresolution wavelet method [19] and the precorrected FFT method [18][6] have been applied to the calculation of drag forces in Stokes flow. Accelerated boundary element solvers that use specialized Green's functions to save time and memory have also been developed [45] and were presented in Chapter 4. These solvers produce the instantaneous force on the surface of objects immersed in Stokes flow given the velocity of those objects at a given time. In the following we use the solver described in Chapter 4 but the methods that are presented in this chapter can be integrated with any other boundary element solver for Stokes flow.

For a matter of notational convenience, let $\mathbf{F}$ describe the vector of forces on the panels representing the objects in the fluid and let $\mathbf{V}$ represent the vector of velocities on these panels, possibly weighted or integrated over the panel areas or multiplied by test functions, depending on the testing scheme that is used. Moreover, let $\mathbf{V}^\infty$ a vector of background flow velocities also evaluated in the object panels according to the discretization of the boundary integral equations that is being used. Let $\mathbf{X}$ represent the configuration of the objects in the flow. For the case of rigid body motion $\mathbf{X}$ is a vector containing the position and orientation of each object in the flow. In the following we will assume that, a boundary element solver exists that can solve the equation

$$\mathbf{G}(\mathbf{X})\mathbf{F} = -\mathbf{V} + \boldsymbol{\gamma}(\mathbf{X})\mathbf{V}^\infty \tag{6.11}$$

where $\boldsymbol{\gamma}$ is a matrix that maps the background velocity vector $\mathbf{V}^\infty$ to the appropriate values on the right hand side of (6.11).

6.2 Time constants and scaling

It would seem that, to simulate the motion of objects in Stokes flow one would simply need to solve (6.11) for $\mathbf{F}$, calculate the force on each object and use these forces to update the velocity and position of the objects by integrating the equations of motion. However, for small length scales, such as those present in MEMS and microfluidic devices, the ratio of the drag forces and the mass of the bodies is such that the time constant associated with transferring momentum between an object and the fluid is very small. In this section we illustrate the small time scale issue by using the trivial example of Stokes flow on a sphere moving in infinite Stokes flow with uniform velocity $\mathbf{v}^\infty$.

6.2.1 Stiffness

Consider a sphere of radius a and density ρ moving with linear velocity v in an infinite fluid domain with viscosity μ and a background velocity v^∞ along the direction of v. Due to symmetry, the position of the sphere, x, and its velocity, v, can be represented by scalar values. In these conditions, the Stokes drag on the sphere is $-6\pi\mu a(v - v^\infty)$ and the position and the velocity of the sphere satisfy

$$\begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & -\frac{9}{2}\frac{\mu}{\rho a^2} \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{9}{2}\frac{\mu}{\rho a^2} \end{bmatrix} v^\infty \tag{6.12}$$

The matrix $\mathbf{A}$ has eigenvalues $\lambda_0 = 0$ and $\lambda_l = \frac{9}{2}\frac{\mu}{\rho a^2}$ that correspond to time constants of $\tau_0 = \infty$ and $\tau_l = \frac{2}{9}\frac{\rho a^2}{\mu}$. Given an initial position x_0 and the initial velocity v_0 the solution of (6.12) is

$$\begin{aligned} v(t) &= v^\infty + (v_0 - v^\infty)\exp(-t/\tau_l) \\ x(t) &= x_0 + v^\infty t - (v_0 - v^\infty)\tau_l(\exp(-t/\tau_l) - 1) \end{aligned} \tag{6.13}$$

For a sphere with a 10μm radius with density $\rho = 10^3$kg/m^3 in a fluid with viscosity $\mu = 8.9 \times 10^{-4}$Pa.s the time constant τ_l is roughly 25μs. On the other hand, we are interested in tracking the motion of the sphere, the time scale of interest is determined by ration between v^∞ and the lengthscale of interest for the device, i.e. the convective time scale, which in microfluidic devices can be on the order of seconds or minutes. It is clear from (6.12) and (6.13) that the small timescale τ_l exists due to the relation between the velocity of the sphere, its mass, and fluid drag force on its surface and that this value is not dependent on the actual position of the object.

The position and velocity in (6.13) are the superposition of a smooth, steady state solution, and a rapidly decaying homogeneous solution. Except for an initial transient, while the velocity of the sphere does not yet match the velocity of the background flow, the behavior of (6.13) is very smooth. However, it will be shown in Section 6.3 that, regardless of how close v is to v^{∞}, if an explicit time integration algorithm is used, very small time steps are needed to maintain stability.

The existence of a small time scale associated with the transfer of momentum from a body to the surrounding Stokes flow is not limited to the case of a sphere translating in infinite flow. The rate at which angular momentum is transfered between a rotating sphere and a background flow is also very high. Moreover, if the sphere, or other object, is close to any other fixed structure in the fluid, such as the bottom of a microfluidic channel, the drag force will increase and the time constant associated with the momentum transfer due to Stokes drag will decrease. In a quiescent fluid, the Stokes drag force will always oppose the motion of the objects moving in the fluid, extracting momentum from the objects and dissipating it in the fluid, or transmitting it to other objects in the fluid.

If only a single object is present in the fluid, the work done by the object on the fluid is entirely dissipated, the fluid does not accumulate kinetic energy. This is demonstrated in Appendix A, at the end of this chapter.

Note on stiffness and volume discretization methods

In the finite element method as well as in the finite difference and finite volume methods, the fluid domain is discretized into a set of small elements. The scaling of volume and area of these elements is such that the time constant associated with momentum transfer through viscous forces on the surface of each element is small, even compared to the time scale of momentum transfer from the bodies immersed in the fluid (which are usually larger than the fluid elements). Therefore, explicitly discretizing the fluid volume and accounting for the momentum in its interior will generate yet another, smaller, lengthscale that the time integration algorithm must deal with.

6.3 Time stepping schemes

This section contains a brief review of a few basic time stepping schemes illustrating the issues associated with numerically integrating the equations of motion (6.12) for a sphere

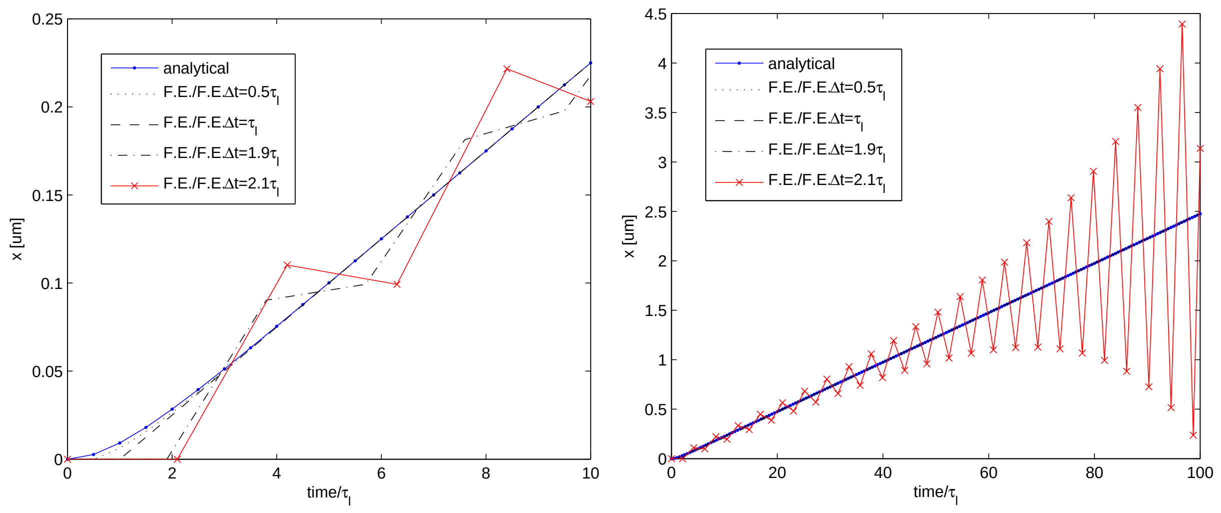


Figure 6-6: *Forward Euler, explicit, numerical time integration of equations of motion for a perfectly buoyant sphere with 10μm radius translating in a fluid with viscosity $\mu = 8.9 \times 10^{-4}$Pa.s and density 1000kg/m^3 domain with a background velocity of $v_\infty = 1$mm/s.*

translating in a fluid with with background flow v^∞.

6.3.1 Forward Euler

If (6.12) is integrated using the Forward Euler method with a time step Δt, the following iteration is produced

$$\begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 - \Delta t / \tau_l \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix}. \tag{6.14}$$

This iteration is unstable for $\Delta t > 2\tau_l$. Therefore, regardless of accuracy concerns, simulating the motion of a 10μm sphere with density 1000kg/m^3 in a fluid with viscosity $\mu = 8.9 \times 10^{-4}$Pa.s, would require taking time steps smaller than 50μs. The stability threshold is independent of v^∞.

The position of the sphere was calculated using (6.14) with an initial velocity of $v = 0$m/s using different values of Δt. The results are illustrated in Figure 6-6, together with (6.13) clearly demonstrating the issue with stability.

6.3.2 Velocity implicit method

Since the small time scale τ_l is associated with the update of the velocity, it is reasonable to expect that using an implicit time integration scheme for updating the velocity update can result in a stable scheme for integrating the equations of motion (6.12). The simplest form of the velocity implicit method replaces the Forward Euler update for the velocity by a Backward Euler update

$$\begin{bmatrix} 1 & 0 \\ 0 & 1+\Delta t/\tau_l \end{bmatrix} \begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta t/\tau_l \end{bmatrix} v^\infty$$

that yields the iteration

$$\begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & (1+\Delta t/\tau_l)^{-1} \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \begin{bmatrix} 0 \\ (1+\Delta t/\tau_l)^{-1}\Delta t/\tau_l \end{bmatrix} v^\infty \tag{6.15}$$

which is stable for all Δt. This method is also known as the symplectic Euler method (see [46], chapter 5).

A more accurate scheme can use the trapezoidal rule for the position update

$$\begin{bmatrix} 1 & -\frac{\Delta t}{2} \\ 0 & 1+\Delta t/\tau_l \end{bmatrix} \begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \frac{\Delta t}{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta t/\tau_l \end{bmatrix} v^\infty$$

yielding the iteration

$$\begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \frac{\Delta t}{2}(1+(1+\Delta t/\tau_l)^{-1}) \\ 0 & (1+\Delta t/\tau_l)^{-1} \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \begin{bmatrix} (1+\Delta t/\tau_l)^{-1}\Delta t^2/2\tau_l \\ (1+\Delta t/\tau_l)^{-1}\Delta t/\tau_l \end{bmatrix} v^\infty \tag{6.16}$$

which is stable for any Δt.

A more accurate scheme can use the trapezoidal rule for the position update and the velocity

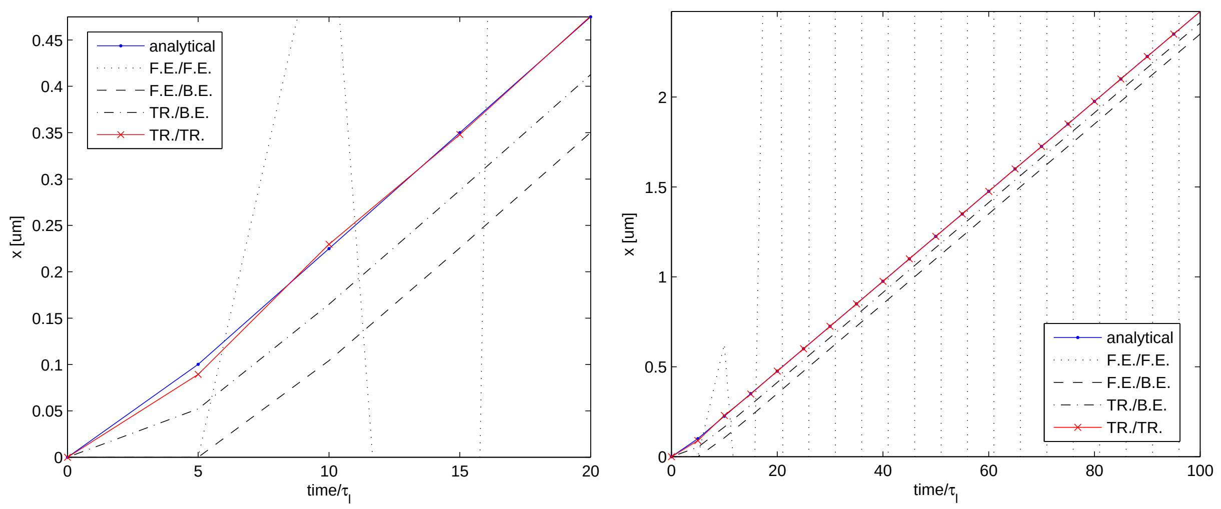


Figure 6-7: *Numerical time integration of equations of motion for a perfectly buoyant sphere with 10μm radius translating in a fluid with viscosity $\mu = 8.9\times10^{-4}$Pa.s and density 1000kg/m^3 domain with a background velocity of $v_\infty = 1$mm/s. F.E. stands for Forward Euler; B.E. for Backward Euler and TR for the trapezoidal method. For example F.E./B.E. means that the Forward Euler method was used to discretize the position update equation and Backward Euler was used to discretize the velocity update equation.*

update.

$$\begin{bmatrix} 1 & -\frac{\Delta t}{2} \\ 0 & 1+\frac{\Delta t}{2}\frac{1}{\tau_l} \end{bmatrix} \begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \frac{\Delta t}{2} \\ 0 & 1-\frac{\Delta t}{2}\frac{1}{\tau_l} \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta t/\tau_l \end{bmatrix} v^\infty$$

yielding the iteration

$$\begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \frac{\Delta t \tau_l}{\tau_l+\Delta t/2} \\ 0 & \frac{2\tau_l-\Delta t}{2\tau_l+\Delta t} \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \begin{bmatrix} (1+\Delta t/\tau_l)^{-1}\Delta t^2/2\tau_l \\ (1+\Delta t/\tau_l)^{-1}\Delta t/\tau_l \end{bmatrix} v^\infty \qquad (6.17)$$

which is also stable for any Δt.

The position of the sphere used in the previous example was calculated using (6.14), (6.15), (6.16), and (6.17) with a time step of $\Delta t = 5\tau_l$. The results are illustrated in Figure 6-7 where it can be observed that (6.15), (6.16), and (6.17) are stable while (6.14) is clearly unstable for $\Delta t = 5\tau_l$.

The local position and velocity truncation errors for the first step of (6.14), (6.15), (6.16) was calculated for several Δt and is illustrated in Figure 6-8 where it can be observed that

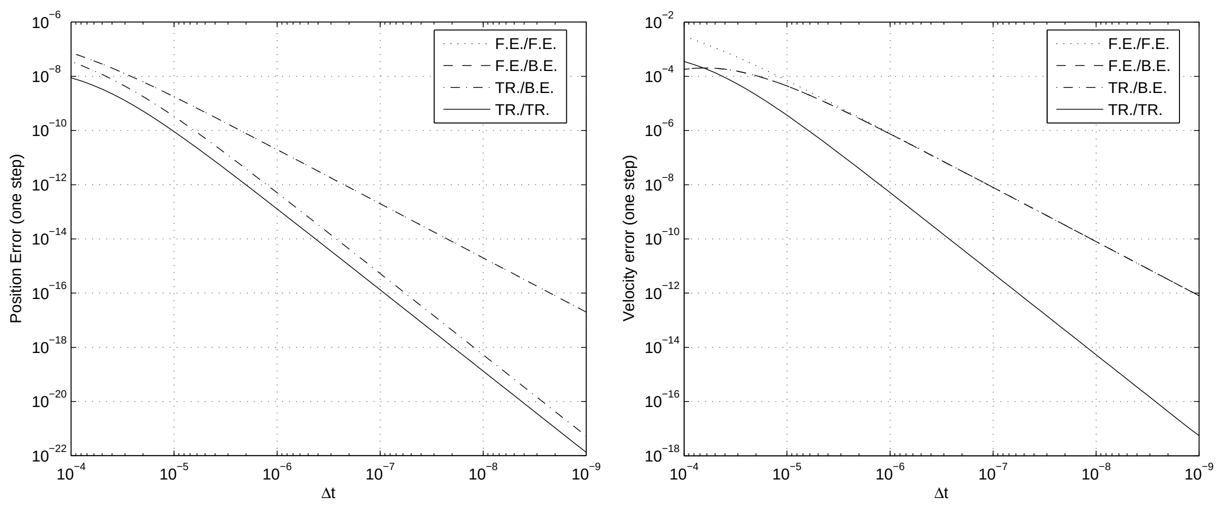


Figure 6-8: *Truncation error for first time step of each numerical integration scheme as a function of the time step. The figure on the left represents the truncation error for the position; the figure on the right represents the truncation error for the velocity.*

the position and velocity truncation error for (6.14), (6.15) are $O(\Delta t^2)$, that the velocity truncation error for (6.15) is also $O(\Delta t^2)$ and that the position truncation error for for (6.16) and (6.17) is $O(\Delta t^3)$.

There exists an enormous number of integration schemes for ordinary differential equations [46]. The purpose of this chapter is not to review all the possible time integration schemes but to show how they can be coupled efficiently with a boundary element solver for Stokes flow.

6.4 Coupling the Stokes BEM solver and rigid body dynamics

In this section a stable time integration algorithm that couples the Stokes drag forces calculated using a boundary element solver with a rigid body model for the objects in the flow is presented. In the following, we consider that objects in the flow are rigid objects of uniform density that are described by a mesh of flat panels. To model more interesting behavior of objects such as cells and vesicles in the flow it is necessary to account for the properties of the membranes of these objects (for details on membrane and capsule simulation see [1],

[47] and [26]). However, the focus of this chapter is to introduce a technique to couple a boundary element solver and a semi-implicit time integration scheme and our discussion is limited to rigid body dynamics. Extending the results in this section to the mode general case of the motion of elastic membranes and shells in flow is left as future work.

First, to introduce some notation and to define relations between the rigid body state and the panel velocities and position a brief review of some basic principles of rigid body motion is presented. Then, in Section 6.4.1, a simple explicit time stepping scheme is presented. Finally, in Section 6.4.2, a more stable velocity-implicit time stepping scheme is presented.

Rigid body motion and notation

To calculate the motion of a rigid body one can use the conservation of linear momentum $\mathbf{P} = m\mathbf{v}$ and angular momentum $\mathbf{L} = \mathbf{I}\boldsymbol{\omega}$, where m is the mass of the body, $\mathbf{v}$ is its linear velocity, $\mathbf{I}$ is its inertia tensor and $\boldsymbol{\omega}$ is its angular velocity. In the following, for convenience, we also use $\mathbf{q} = [\mathbf{v}^T \ \ \boldsymbol{\omega}^T]^T$. The spatial configuration of a rigid body is determined by the position of its center of mass, $\mathbf{x}$ and by its orientation, which can be represented by a rotation matrix, $\mathbf{R}$, mapping positions in the body's local coordinate system to the global coordinate system, or by unit quaternion $\mathbf{Q}$ [48]. In the following, $\mathbf{X}$ is used to represent the set of positions and orientations of the rigid bodies in the flow, i.e. the current configuration of the system. The body's inertia tensor is a function of its orientation and is given by $\mathbf{I}(\mathbf{X}) = \mathbf{R}\mathbf{I}_0\mathbf{R}^T$ where $\mathbf{I}_0$ is the inertia tensor in the body's local coordinate system. The center of mass and inertia tensor $\mathbf{I}_0$ for any constant density object defined by a flat panel mesh can be calculated using the algorithm proposed in [49].

Linear and angular momentum conservation state that $\dot{\mathbf{P}} = \mathbf{f}$ and $\dot{\mathbf{L}} = \mathbf{T}$ where $\mathbf{f}$ and $\mathbf{T}$ are the total force and torque applied to the object. From momentum conservation it follows (see [50]) that the velocity and angular velocity of a rigid body satisfy the following equation

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{1}_3 m^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}(\mathbf{X})^{-1} \end{bmatrix}}_{\mathbf{M}(\mathbf{X})^{-1}} \begin{bmatrix} \mathbf{f} \\ \mathbf{T} + \mathbf{L} \times \boldsymbol{\omega} \end{bmatrix} \tag{6.18}$$

where $\mathbf{1}_3$ is the 3 by 3 identity matrix. For spherical objects $\mathbf{L} \times \boldsymbol{\omega}$ is zero.

Given a rigid body described by a set of n_p flat panels, if the Stokes drag force is given by $\mathbf{F} \in R^{3n_p}$ the Stokes drag force on the object is

$$\mathbf{f}_S = \sum_{k=1}^{n_p} a_k \mathbf{F}_{3(k-1)+(1:3)} \tag{6.19}$$

where a_k is the area of panel k. The total torque due to Stokes drag on the object is

$$\mathbf{T}_S = \sum_{k=1}^{n_p} a_k (\mathbf{x}_k - \mathbf{x}) \times \mathbf{f}_k \tag{6.20}$$

where $\mathbf{x}_k$ is the centroid of panel k. This projection operation, from the panel forces $\mathbf{F}$ to the total force (6.19) and total torque (6.20), is represented in matrix form by a $3n_p$ by 6 matrix $\mathbf{B}(\mathbf{X})$ such that

$$\begin{bmatrix} \mathbf{f}_S \\ \mathbf{T}_S \end{bmatrix} = \mathbf{B}^T(\mathbf{X})\mathbf{F}.$$

where $\mathbf{B}(\mathbf{X})$ is a function of the orientation of each object.

To calculate the Stokes drag on an object using a boundary element solver with a collocation testing scheme, represented generically by (6.11), a vector $\mathbf{V} \in R^{3n_p}$ with the velocity on the centroid of each panel is needed. The vector $\mathbf{V}$ is given by

$$\mathbf{V}_{3(k-1)+(1:3)} = \mathbf{v}_k = \boldsymbol{\omega} \times (\mathbf{x}_k - \mathbf{x}) + \mathbf{v}.$$

This expansion operation is represented in matrix form by a $3n_p$ by 6 matrix $\mathbf{A}(\mathbf{X})$ such that

$$\mathbf{V} = \mathbf{A}(\mathbf{X}) \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} = \mathbf{A}(\mathbf{X})\mathbf{q}$$

If the BEM solver is using a Galerkin testing scheme, instead of the vector of velocities on all panels it will need a vector of fluxes $\hat{\mathbf{V}}$. The vector of fluxes $\hat{\mathbf{V}}$ is given by

$$\hat{\mathbf{V}} = \mathbf{B}(\mathbf{X}) \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} = \mathbf{B}(\mathbf{X})\mathbf{q}$$

6.4.1 An explicit coupled solved

Using the notation defined in the previous section, the following system of equations representing an explicit time integration algorithm coupling the boundary element solver to rigid body dynamics can be written

$$\begin{bmatrix} \mathbf{1}_6 & -\Delta t \mathbf{M}^{-1}(\mathbf{X}_k)\mathbf{B}(\mathbf{X}_k)^T \\ \mathbf{0} & \mathbf{G}(\mathbf{X}_k) \end{bmatrix} \begin{bmatrix} \mathbf{q}_{k+1} \\ \mathbf{F} \end{bmatrix} =$$
$$\begin{bmatrix} \mathbf{1}_6 + \Delta t \mathbf{K}(\mathbf{X}_k) & \mathbf{0} & \Delta t \mathbf{M}^{-1}(\mathbf{X}_k) \\ -\mathbf{A}(\mathbf{X}_k) & \gamma(\mathbf{X}_k) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{q}_k \\ \mathbf{V}_\infty \\ \mathbf{f}_{\text{ext}} \\ \mathbf{T}_{\text{ext}} \end{bmatrix} \tag{6.21}$$

where $\mathbf{f}_{\text{ext}}$ and $\mathbf{T}_{\text{ext}}$ represent the total externally applied force and torque and

$$\mathbf{K}(\mathbf{X}_k) = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}(\mathbf{X}_k)^* \end{bmatrix} \text{ and } \mathbf{L}^* = \begin{bmatrix} 0 & -L_3 & L_2 \\ L_3 & 0 & -L_1 \\ -L_2 & L_1 & 0 \end{bmatrix}.$$

The system of equations (6.21) is a block upper triangular system of equations that can be solved for $\mathbf{F}$ by using a boundary element solver. The resulting $\mathbf{F}$ is then used to produce $\mathbf{q}_{k+1}$, which in turn is used to calculate $\mathbf{x}_{k+1}$ and $\mathbf{Q}_{k+1}$ according to

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \Delta t(\mathbf{v}_k + \mathbf{v}_{k+1})/2 \\ \mathbf{Q}_{k+1} &= (\Delta t/2\boldsymbol{\omega}_{k+1}) * (\Delta t/2\boldsymbol{\omega}_k) * \mathbf{Q}_k \end{aligned} \tag{6.22}$$

yielding $\mathbf{X}_{k+1}$. Note that, for numerical stability, the orientation of each object is represented by a unit quaternion that is updated using finite rotations, instead of incrementally.

6.4.2 Velocity implicit coupled solver

From the results and examples presented in sections 6.3 and 6.2, it is clear that a stable solver will require that the Stokes drag force be calculated in an implicit manner consistent with the velocity updating scheme. An implicit scheme can be obtained by modifying (6.21) such that $\mathbf{F}$ depends not only on $\mathbf{q}_k$ but also on $\mathbf{q}_{k+1}$ i.e.

$$\begin{bmatrix} \mathbf{1}_6 & -\Delta t \mathbf{M}^{-1}(\mathbf{X}_k)\mathbf{B}(\mathbf{X}_k)^T \\ \alpha_0 \mathbf{A}(\mathbf{X}_k) & \mathbf{G}(\mathbf{X}_k) \end{bmatrix} \begin{bmatrix} \mathbf{q}_{k+1} \\ \mathbf{F} \end{bmatrix} = \begin{bmatrix} \mathbf{1}_6 + \Delta t \mathbf{K}(\mathbf{X}_k) & \mathbf{0} & \Delta t \mathbf{M}^{-1}(\mathbf{X}_k) \\ -\alpha_1 \mathbf{A}(\mathbf{X}_k) & \boldsymbol{\gamma}(\mathbf{X}_k) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{q}_k \\ \mathbf{V}_\infty \\ \mathbf{f}_{\text{ext}} \\ \mathbf{T}_{\text{ext}} \end{bmatrix} \tag{6.23}$$

where, for "Backward-Euler" type update, $\alpha_0 = 1$ and $\alpha_1 = 0$ and, for a "Trapezoidal" type update, $\alpha_0 = 1/2$ and $\alpha_1 = 1/2$. Note that, regardless of the value used for α_0 and α_1, the update for the velocity is not entirely implicit because the gyroscopic term $\mathbf{L} \times \boldsymbol{\omega} = (\mathbf{I}\boldsymbol{\omega}) \times \boldsymbol{\omega}$, introduces a non-linear dependence on $\boldsymbol{\omega}$. For example, for a Backward Euler update one would need to solve

$$\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k + \Delta t \mathbf{T}_{k+1} + \Delta t \mathbf{I}_k \boldsymbol{\omega}_{k+1} \times \boldsymbol{\omega}_{k+1}.$$

The system of equations (6.23) is not block upper triangular like (6.21) but it can be solved using GMRES. Alternatively, (6.23) can be reduced to a block upper triangular form by applying a step of Gaussian elimination yielding

$$\begin{bmatrix} \mathbf{1}_6 & -\Delta t\mathbf{M}^{-1}(\mathbf{X}_k)\mathbf{B}(\mathbf{X}_k)^T \\ \mathbf{0} & \mathbf{G}(\mathbf{X}_k) + \alpha_0\Delta t\mathbf{A}(\mathbf{X}_k)\mathbf{M}(\mathbf{X}_k)\mathbf{B}(\mathbf{X}_k)^T \end{bmatrix} \begin{bmatrix} \mathbf{q}_{k+1} \\ \mathbf{F} \end{bmatrix} =$$
$$\begin{bmatrix} \mathbf{1}_6 + \Delta t\mathbf{K}(\mathbf{X}_k) & \mathbf{0} & \Delta t\mathbf{M}^{-1}(\mathbf{X}_k) \\ -\mathbf{A}(\mathbf{X}_k)(\mathbf{1}_6 + \alpha_0\Delta t\mathbf{K}(\mathbf{X}_k)) & \boldsymbol{\gamma}(\mathbf{X}_k) & -\Delta t\mathbf{A}(\mathbf{X}_k)\mathbf{M}^{-1}(\mathbf{X}_k) \end{bmatrix} \begin{bmatrix} \mathbf{q}_k \\ \mathbf{V}_\infty \\ \mathbf{f}_{\text{ext}} \\ \mathbf{T}_{\text{ext}} \end{bmatrix} \tag{6.24}$$

In our implementation use the form (6.24) is used and a solution for

$$\left(\mathbf{G} + \alpha_0\Delta t\mathbf{A}\mathbf{M}^{-1}\mathbf{B}^T\right)\mathbf{F} = -\mathbf{A}(\mathbf{1}_6+\alpha_0\Delta t\mathbf{K})\mathbf{q}_k - \Delta t\mathbf{A}\mathbf{M}^{-1}\left[\mathbf{f}_{\text{ext}}^T\ \mathbf{T}_{\text{ext}}^T\right]^T + \boldsymbol{\gamma}\mathbf{V}_\infty \tag{6.25}$$

is computed using GMRES and the pFFT accelerated representation of $\mathbf{G}$ described in Chapter 4 and a rank 6 update for each rigid object moving in the fluid. Then the calculated force $\mathbf{F}$ is used to calculate $\mathbf{q}_{k+1}$ in a consistent manner that is stable for Δt much larger than τ_l. If $\mathbf{G}$ does not change with $\mathbf{x}$ this scheme is stable for all Δt, if $\mathbf{G}$ changes with $\mathbf{x}$ it might be possible to construct an example where the iteration scheme becomes unstable but we have not encountered any case where this happens. Regardless of stability, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{G}$ and $\mathbf{K}$ are functions of $\mathbf{X}_k$ so the time step Δt is still limited by accuracy concerns i.e. by the requirement that these matrices be a reasonable approximations to their actual values for the path from $\mathbf{X}_k$ to $\mathbf{X}_{k+1}$.

External forces

Many interesting problems involve not only fluid drag forces but also forces such as gravity and electrical forces. Often the relation between the external forces and the mass of the objects that are to be simulated is such that taking a time step considering the acceleration due to these forces, without considering the immediate response of the Stokes drag, would make the simulated object, wrongly, leave the domain of interest. For a simple example illustrating this problem, consider a spherical bead with a diameter $a = 10\mu$m and a density of $\rho_b = 1010$kg/m^3 in an infinite quiescent fluid with density $\rho = 1000$kg/m^3 and viscosity $\mu = 8.9 \times 10^{-4}$Pa.s. The balance between the Stokes drag force and the gravitational force imply that the bead's terminal velocity will satisfy $4/3(\rho_b - \rho)\pi a^3 g - 6\pi\mu a V_z = 0$ i.e. $V_z = 8(\rho_b - \rho)a^2 g/\mu$, where $g = 9.8$m/s^2 is the gravitational acceleration. The analytical solution, for the case where the bead starts from rest is $v(t) = V_z * (1 - \exp(-t/\tau_l))$ where, for this example, $V_z \approx 88.1\mu$m/s and $\tau_l \approx 25\mu$s. For a time step of 0.1s, the analytical solution will move at most 8.1μm, on the other hand, if the acceleration due to g was considered separately and a time step of 0.1s was taken, the bead would have been moved by 485.15μm.

To avoid having to use very small time steps, external forces, $\mathbf{f}_{\text{ext}}$, and torques, $\mathbf{T}_{\text{ext}}$, should be calculated at the beginning of each iteration and the resulting acceleration on the objects in the flow should be considered by the Stokes flow solver so that an appropriate drag force can be calculated as in (6.25). An exception to this rule are user computed constraint forces whose calculation requires access to the total force and torque on each object [51][52][50].

6.5 Interaction with structures

Since the objects moving in the fluid may collide with each other and with fixed structures in the fluid, collisions, friction and contacts must be modeled. To deal with these issues the freely available library ODE [27] was integrated with the simulator.

Before each time step, our time integration algorithm checks for collisions between the objects. To detect collisions and penetrations our algorithm uses the libraries OPCODE or GIMPACT that are associated with the ODE library.

At the beginning of each step **F** is calculated for the candidate time step $t + \Delta t$ by solving (6.25). The force and torque on each object is then calculated from F and is used to generate a candidate state for time $t + \Delta t$.

If a collision was detected at the beginning of the time step, ODE is used to generate a step candidate that integrates the equations of motion considering not only the forces and torques due to fluid drag, calculated from $\mathbf{F}$, and external sources, but also contact, friction and collision forces [27].

If no collision was detected at the beginning of the time step, a candidate state for time $t + \Delta t$ is computed using (6.24) and (6.22).

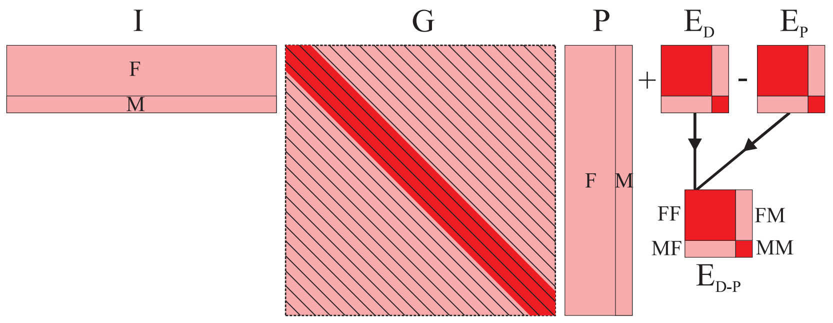
The candidate state is checked for penetrations and contacts. If there is any penetration that exceeds a user-defined limit, the candidate state is rejected and the time step size is reduced. If no excessive penetrations were detected, the candidate state position of each object is compared against a polynomial prediction based on the position at previous time steps; if the difference between the two values exceeds a user-defined threshold the step is rejected. The step size is adjusted using the following criteria [53]

$$\Delta t_{\text{new}} = \Delta t_{\text{old}} \gamma \left(\frac{\text{Error}_{\text{maximum}}}{\text{Error}_{\text{estimate}}} \right)^{n+1}$$

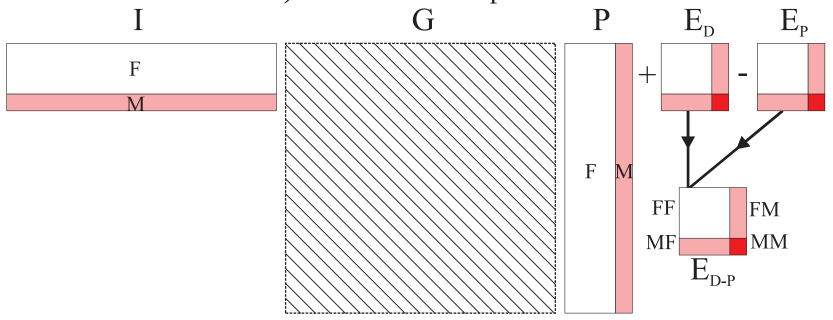
where n is the order of the integration scheme and the polynomial predictor; $\text{Error}_{\text{estimate}}$ is a user defined tolerance (in meter); $\text{Error}_{\text{estimate}}$ is the absolute value of the difference between the predicted position value and the corresponding candidate state position value; and is chosen to be about 0.9 to reduce the number of rejected steps. To avoid very large step variations $\Delta t_{\text{new}}/\Delta t_{\text{old}}$ is constrained to be within 0.5 and 2.

After each time step, the time integration routine calls a visitor functor with the state of each object in the flow and the current simulation time. The visitor functor indicates if the simulation is to continue or if it should be terminated. By default, the visitor does not terminate the simulation and the time integration routine finishes only when the end time specified for the simulation is reached.

Note that when a step is rejected the calculation of $\mathbf{F}(t + \Delta t)$ for the new step size does not require setting up a new Stokes flow boundary element operator; only the rank 6 update operator for each object and the right hand side of (6.25) need to be recalculated.



a) No reuse. Full update.



b) Reusing grid data and precorrection entries between fixed structures.

Figure 6-9: *pFFT matrices and the updates that must be made from iteration to iteration as the time integration algorithm is executed. In the figure above the pink color represents sparse storage or low effort required; red represents high storage or computational effort and white represents no computational effort (due to reuse). In the figure F stands for fixed and M for movable.*

6.6 Updatable solver

A significant part of the computational cost of solving (6.25) is setting up the precorrected FFT representation of $\mathbf{G}$ at each iteration. A large part of this cost is due to the calculation of the nearby interactions and the precorrection matrix entries. However, in the examples that are of interest there are usually a few large fixed structures and one or more smaller moving objects. The interactions between the panels of the fixed structures don't change from iteration from iteration and can be reused. Also, if from iteration to iteration the FFT grid spacing is kept constant and the grid is moved and resized by increments of this grid spacing, the precorrection matrix elements for the fixed structures also don't need to be recalculated. Reusing the nearby and precorrection for the fixed structures from iteration to iteration can have large performance benefits. This updatable solver can be implemented by decomposing the precorrected FFT data structures into a set of block sparse matrices separated by fixed, F, and movable, M, as illustrated in Figure 6-9.

If there are fixed objects in the simulation and the grid spacing is not changed from iteration to iteration, the columns in the projection matrix and the rows in the interpolation matrix associated with the fixed objects don't need to be recalculated. More importantly, the elements of the matrices containing the accurate values for the nearby interactions between elements of the fixed objects, $\mathbf{E}_D$, and the precorrection entries, used to cancel out the inaccurate interactions calculated using the grid, $\mathbf{E}_P$, do not need to be updated. If the grid size doesn't change, the kernel transforms $\mathbf{G}$ can also be reused.

For problems where the substrate is not represented implicitly and the free space Stokes Green's function is used, the nearby interactions between the panels of the same moving rigid object can be reused from iteration to iteration by wrapping the nearby interaction matrix in appropriate rotation operators. However, since the objects are moving along the pFFT grid, the precorrection matrix must still be recalculated. To reuse the previously calculated nearby interactions, the storage for the nearby interactions must be separated from the storage for the precorrection term.

6.7 Results

In this section, the stability and effectiveness of the time integration scheme that was introduced in this Chapter is demonstrated by simulating a set problems involving cell traps. The first set of examples uses a microwell trap and the formulation introduced in Section 6.1.3; the second set of examples compares four models of pachinko cell traps and uses the formulation described in Section 6.1.2 and the Stokes substrate Green's function.

For the examples reported in this section, we used ODE's contact models with `dContactApprox1` and `dContactBounce`, `mu=1`, `bounce=0.5`, `soft_cfm=0` and `soft_erp=0.9`. The relative tolarence for the GMRES linear system solver was set to 10^{-5}. The nullspace was removed from the GMRES search space at each iteration step and a right block diagonal preconditioner was used.

6.7.1 Microwell trap

The approximate formulation presented in Section 6.1.3 was used to simulate the behavior of objects trapped in a microwell cell trap such as those described in [24]. For the examples presented in this section a trap with a diameter of 30μm and a depth of 35μm was used. The fluid density was set to $\rho_f = 1000$Kg/m^3 and the fluid viscosity was set to $\mu = 8.9 \times 10^{-4}$Pa.s. The trap and fluid geometry were represented by a triangular mesh generated using Comsol 3.2. The mesh is composed of 1345 panels for the hole walls, 1012 panels for the substrate and 3595 panels for the fluid boundaries. The discretized geometry, and a sample trajectory for a bead moving in the trap, are presented in Figure 6-10. The bead, with density ρ, is represented by an icosphere with 1280 panels. The velocity on the fluid boundary was set to that of a fully developed rectangular channel flow for a 3mm wide and 200μm high channel with a flow rate of F. To study the effect of changing the flow rate F and the bead density ρ on the trapping behavior and to analyze the performance of the transient solver, the flow rate F was set to the values of 100pL/s, 200pL/s and 400pL/s while the bead density ρ was set to 1000kg/m^3, 1010kg/m^3, 1050kg/m^3 and 1100kg/m^3 generating a set of 12 simulations. In each case the bead started from rest, in the trap, at the position $x = -5\mu$m, $y = 0\mu$m and $z = -10\mu$m. The simulations were terminated at time 60s or when a bead escaped the trap and reached the boundary of the computational domain.

Simulation results

The results for sweeping the bead density with a fixed flow rate are illustrated in Figure 6-11; the results for fixing the bead density and sweeping the flow rates are presented in Figure 6-12. The trajectories for the lower flow rates are shorter because the simulations were ran to the same end time. From both figures it is clear that, as expected, heavier beads get captured more easily and that higher flow rate can release lighter beads. However, it is also clear from the figures that the beads trajectories can be somewhat complicated and that they depend strongly on the flow rate and on the bead density.

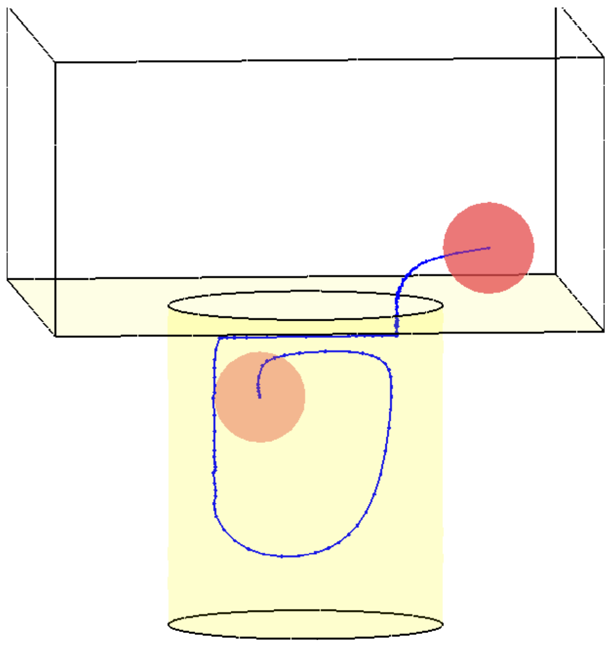
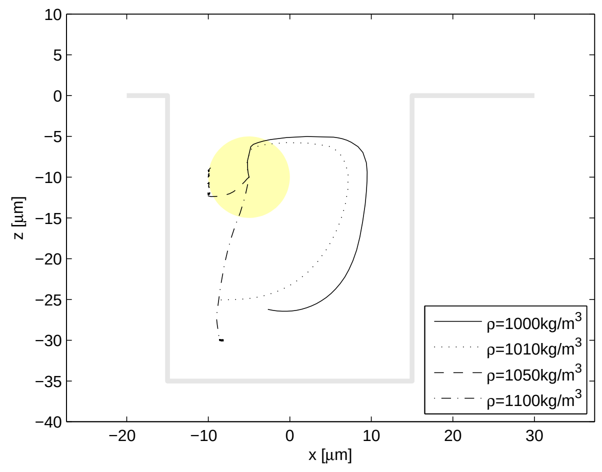
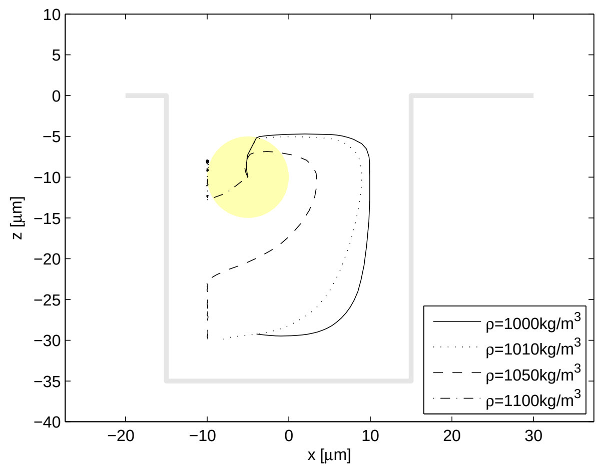


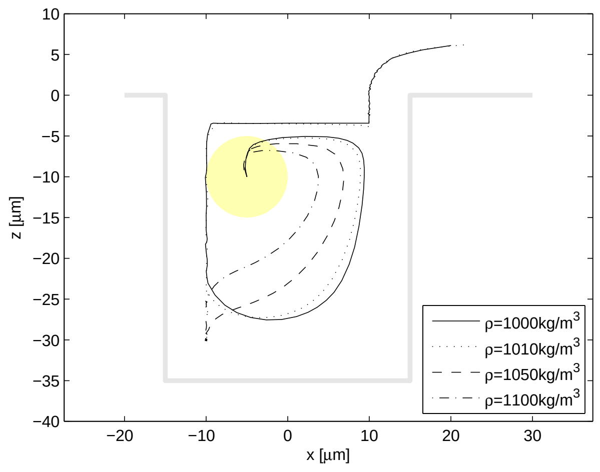
Figure 6-10: *Trajectory of a spherical bead moving inside, and then escaping, a cylindrical microwell with a diameter of 30μm and a depth of 35μm. The bead and fluid density were set to $\rho = 1000$kg/m^3 and the fluid velocity on the boundaries of the fluid domain was set to the profile of a fully developed rectangular channel flow for a 3mm wide and 200μm high channel with a flow rate of 400pL/s. The fluid viscosity was set to $\mu = 8.9 \times 10^{-4}$Pa.s. In the figure the bead is drawn at its initial position, inside the trap, and at its final position, outside the trap. The surface of the microwell and the substrate, where the fluid velocity is zero due to the no-slip boundary condition, is colored light yellow.*



$F = 100\text{pL/s}$



$F = 200\text{pL/s}$



$F = 400\text{pL/s}$

Figure 6-11: *Each figure illustrates the trajectory of the center of mass of a bead for a given flow rate F and for a set of bead density values. The bead is moving inside a cylindrical microwell with a diameter of 30μm and a depth of 35μm (the side view of the trap walls is depicted in light gray). The sphere of radius 5μm, in light yellow, started its motion from rest.*

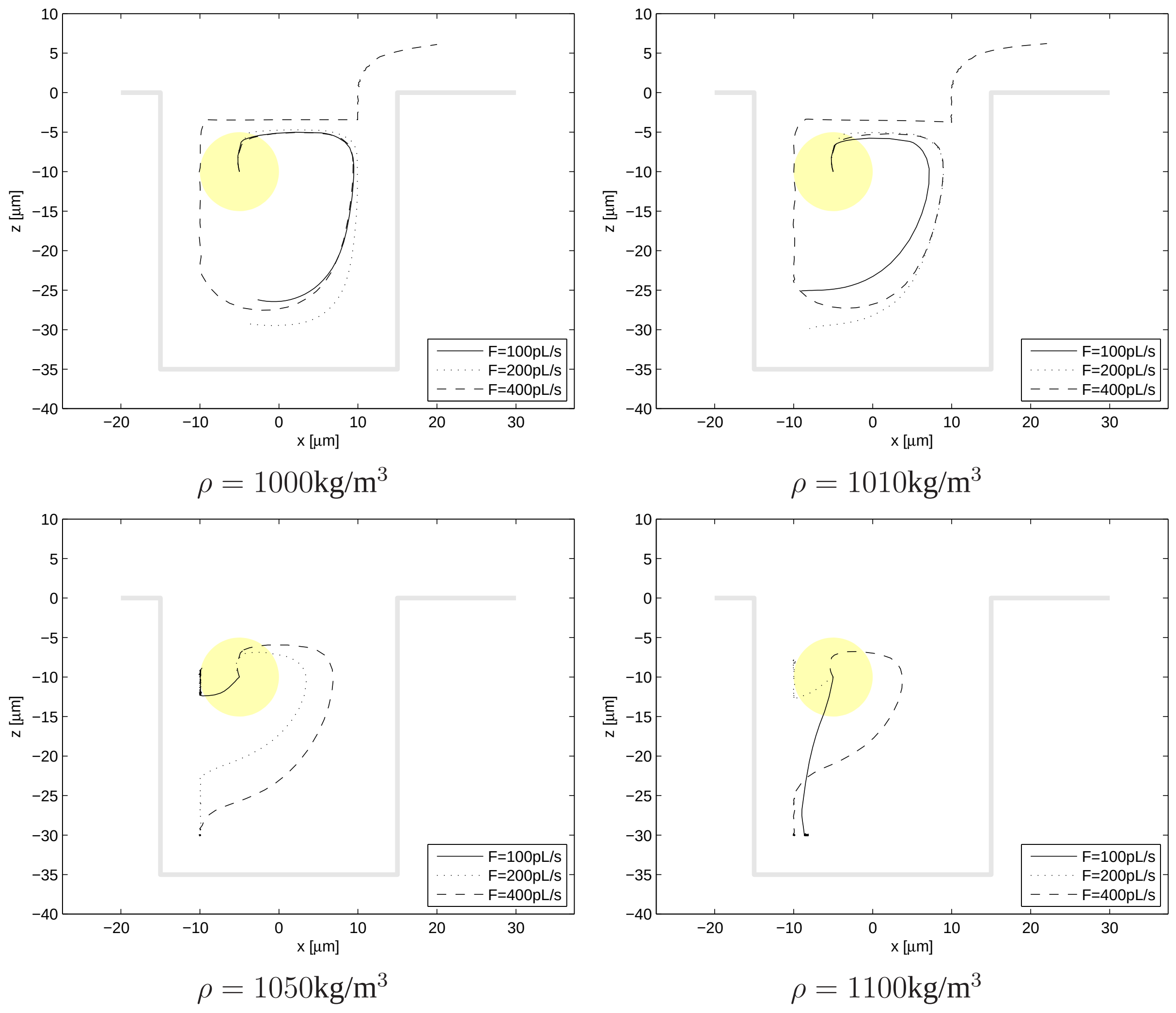


Figure 6-12: *Each figure illustrates the trajectory of the center of mass of a bead for a given bead density ρ and for a set of flow rates F. The bead is moving inside a cylindrical microwell with a diameter of 30μm and a depth of 35μm (the side view of the trap walls is depicted in light gray). The sphere of radius 5μm, in light yellow, started its motion from rest.*

Performance analysis and solver behavior

The 12 simulations were ran on an Intel Xeon 3GHz workstation with 2Gb of RAM and took from 40 minutes to 3 hours to run, depending on the trajectory followed by the bead and the number of collisions that occurred. For each simulation, the precorrected FFT solver used a 48 by 48 by 48 FFT grid and a maximum of 227Mb of memory.

The median time for setting up the precorrected FFT solver was 9.6s. The maximum time for setting up the precorrect FFT solver was 67.2s, corresponding to the first iteration, when the interactions and precorrection terms between the nearby fixed panels are calculated. A histogram with the distribution of setup times for generating the step candidates is presented in Figure 6-13 clearly illustrating the performance benefits due to the selective update of the precorrected FFT data structures. Figure 6-13 also presents a histogram for the GMRES solve times. The median time for solving (6.25) using GMRES was 22.3s; a smaller value might have been obtained by using a coarser FFT grid, at the cost of a larger number of nearby interactions. The median time for generating a step candidate was 33.49s.

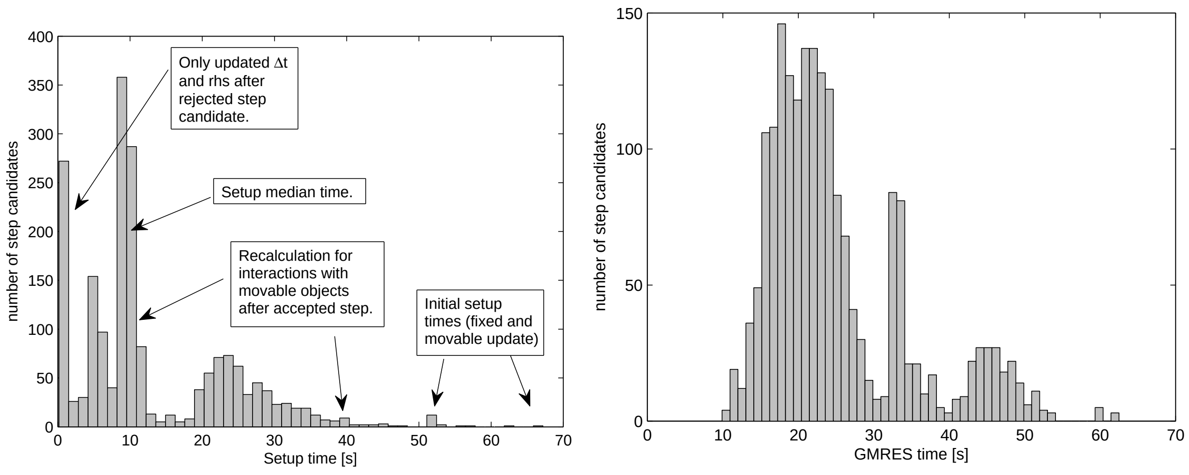


Figure 6-13: *Histogram for the precorrected FFT setup time, on the left, and for the GMRES solve time, on the right, for each step candidate. The figures illustrate compounded results from 1951 step candidates, produced by the 12 simulations that were performed for the bead density and flow rate sweep for the microwell.*

It was observed that, as expected, the step rejection rate and the median step size taken by the time integration algorithm were greatly influenced by the presence or absence of collisions. In the presence of collisions a much larger number of step rejections occurred due to excessive penetration or to excessive integration error. Moreover, as illustrated in Figure 6-14, in the presence of collisions and step rejections, the median step size for the simulations decreases substantially.

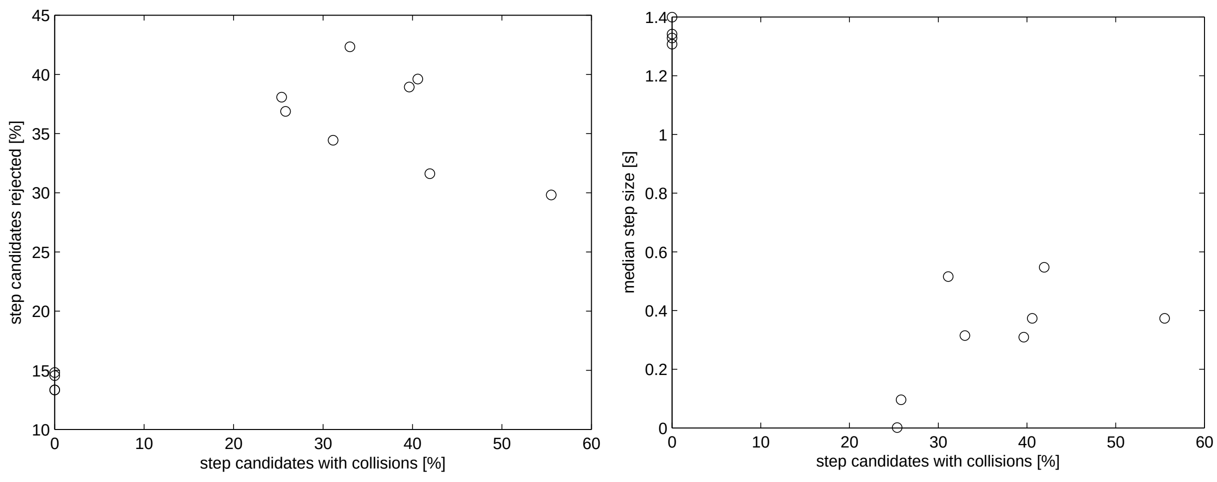


Figure 6-14: *Effect of collisions on step candidate rejection rate, on the left, and on the time integration step size, on the right. The figures illustrate compounded results from 1951 step candidates, produced by the 12 simulations that were performed for the bead density and flow rate sweep for the microwell.*

A larger number of step candidate rejections and the use of smaller step sizes leads to larger run times for the simulations as illustrated in Figure 6-15.

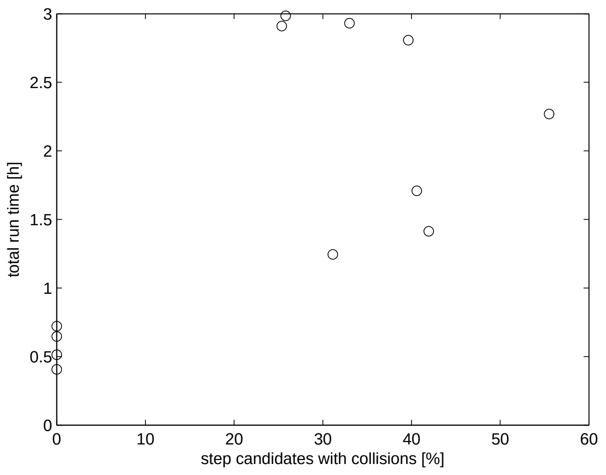
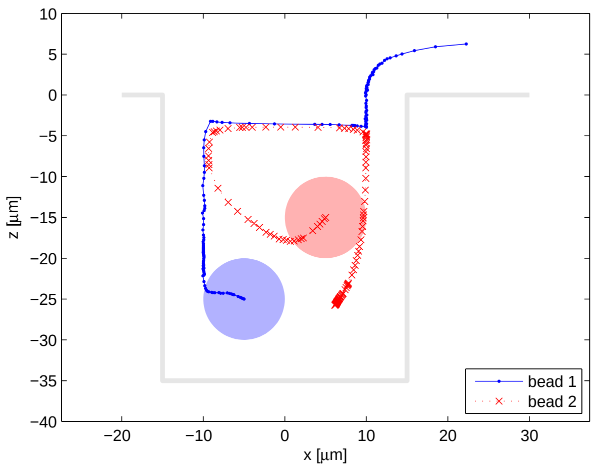
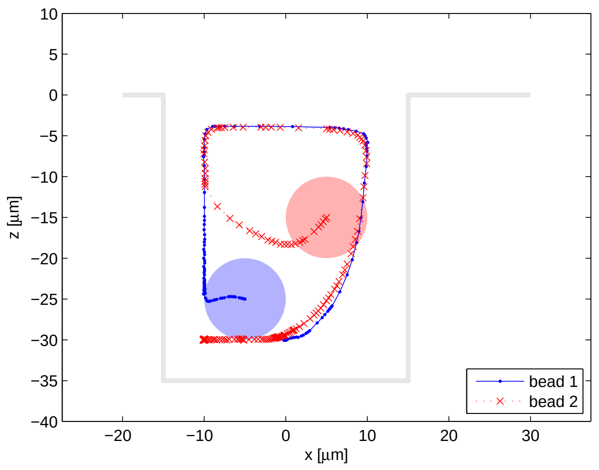


Figure 6-15: *Effect of collisions on the simulation run time. The results in this figure correspond to the 12 simulations that were performed for the bead density and flow rate sweep for the microwell.*



a) $\rho = 1010\text{Kg/m}^3$, $F = 400\text{pL/s}$



b) $\rho = 1025\text{Kg/m}^3$, $F = 400\text{pL/s}$

Figure 6-16: *Trajectory of a pair of spherical beads moving inside a cylindrical microwell with a diameter of 30μm and a depth of 35μm. The fluid density was set to ρ=1000Kg/m^3 and the fluid viscosity was set to $\mu = 8.9 \times 10^{-3}$. Each bead has a diameter of 10μm and was at rest in the beginning of the simulation. The surface of the microwell and the substrate, where the fluid velocity is zero due to the no-slip boundary condition, are depicted in light gray.*

Pair of beads in a microwell

Simulating the trapping of two of more beads is no more complicated than simulating the trapping of a single bead. However, the presence of more than one bead in the trap can lead to a larger number of collisions that will result in smaller time steps and longer simulation run times. The simulation of the pair of beads with $\rho = 1010\text{kg/m}^3$, illustrated in Figure 6-16, took 4.4 hours to run and used 263.8Mb of memory on an Intel Xeon 3GHz workstation with 2Gb of RAM. In this simulation the beads initially collide, one of the beads rotates around in the trap, collides with the trap wall and is released. The simulation of the pair of beads with $\rho = 1050\text{kg/m}^3$ took 4.6 hours to run until it reached a state where the pair of beads was trapped, lying on the bottom of the hole at the simulation time of 60s, as illustrated in Figure 6-16. However, after this state was reached, the simulation continued on with smaller time steps and many collisions; at the simulation time of 90s the simulation had ran for roughly 11 hours taking very small time steps as the two beads collided with each other and with the bottom and side of the trap.

The small time step issues associated with dealing collisions and contacts seem to suggest that it would be useful to find a more efficient way to deal with persistent contacts, such as those that occur when a bead is trapped. However, delving more deeply into the simulation of contacts, collisions and friction is not the focus of this thesis and finding a freely available implementation of the collision detection algorithms or the rigid body dynamics library that is more robust than the ODE package [27] has proved to be quite hard.

6.7.2 Pachinko trap

In this section we present simulation results for protruding cell trap geometries such as those described in [21] using the boundary integral formulation presented in Section 6.1.2. This formulation uses the Stokes substrate Green's function which has the advantage that it only requires discretizing the structures above the substrate as illustrated in Figure 6-17. In the following we present results obtained by using four different trap geometries and different settings for the background flow velocity and for the bead density. The four trap geometries are illustrated in Figure 6-18 where it can be observed that the shape of the trapping region itself was kept constant while the shape of the trap support was changed. A triangular mesh for each trap model was generated using Comsol 3.2: trap model 1 is represented by 2810 panels; trap model 2 by 1820 panels; trap model 3 by 2400 panels; and trap model 4 is represented by 2736 panels. All the traps models are 20μm high. The simulations were ran with a bead represented by an icosphere with 1280 triangular panels. The simulations were terminated when the simulation time reached 15s.

The following observations were made. In the absence of an effective gravitational force, i.e. when the bead density was set to the same value as the fluid density, changing the flow rate does not change the trajectory of the beads. This result is illustrated in Figure 6-19a) and is expected because, in the absence of gravity, due to the linearity of the Stokes equations, multiplying the flow rate by a given factor changes the drag forces and the accelerations by the same factor and hence the objects follow the same trajectory except they do so at a velocity that is multiplied by that same factor. If the bead density is set to a value different than that of the fluid density, the gravity has an effect on the trajectory and, given the initial position of the bead, the relation between the bead density and the flow rate greatly influences if the bead will be captured or if it will escape. The effect of the

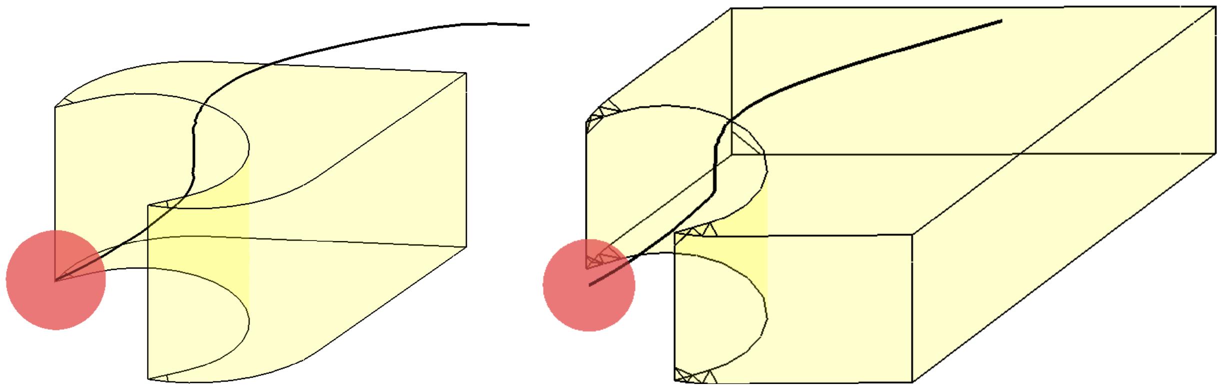


Figure 6-17: *Trajectory of a bead released near a pachinko type trap. On the the trap model is 1 and the bead density was 1050kg/m*^{3}*; on the right the trap model is 2 and the bead density was 1050kg/m*^{3}*. Note that the edges on the corners on the structures are just an artifact of the algorithm used to render the figures.*

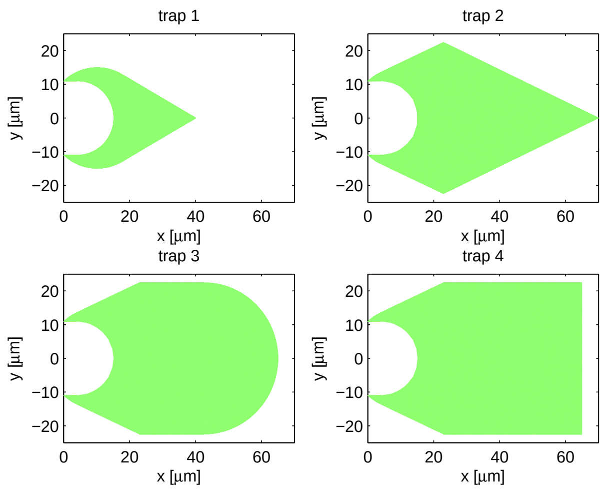
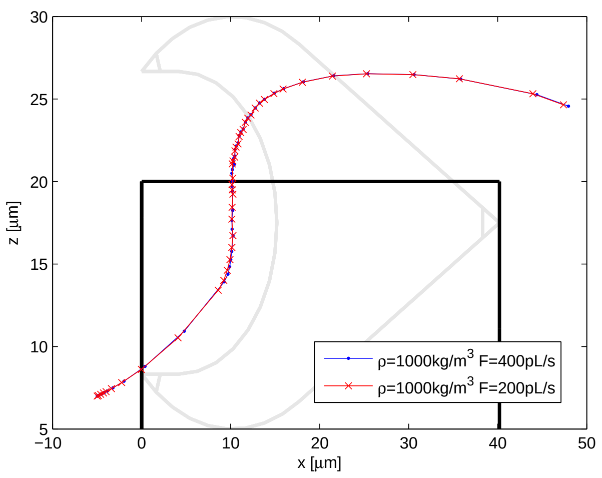


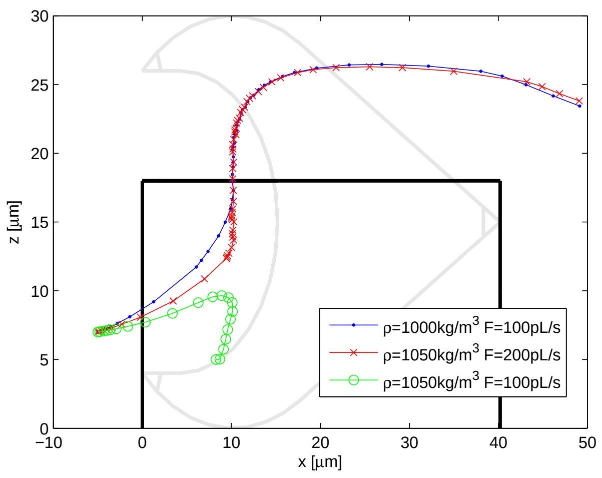
Figure 6-18: *Top view of the 4 pachinko type cell traps used to demonstrate the time domain simulator. Each trap protrudes 20*μ*m above the substrate.*

flow rate on the trapping behavior is illustrated in Figure 6-19b) where a bead with density $\rho = 1050$kg/m^3 is captured with a flow rate of 100pL/s while it escapes for a flow rate of 200pL/s.

Another interesting observation that was made from our numerical experiments was that changing the trap model by changing the shape of the trap support while keeping the actual trapping region the same does not significantly change the trajectory of the bead. This is



a) perfectly buoyant bead

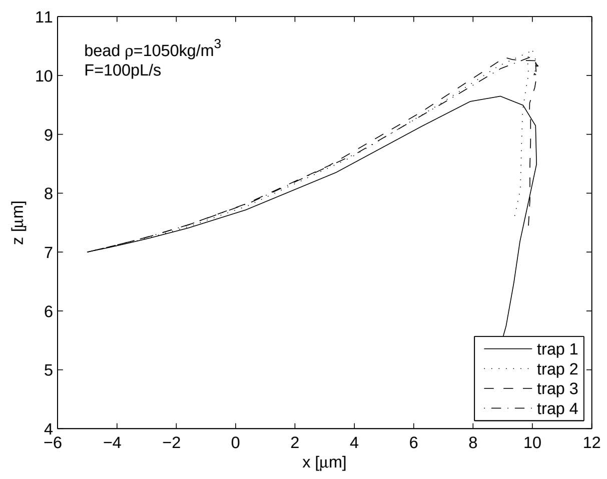


b) slightly heavier bead

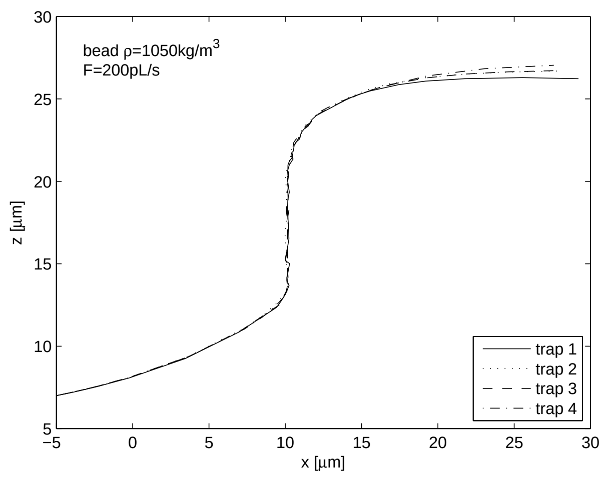
Figure 6-19: *Trajectory of the center of mass of a bead released near a model 1 pachinko trap. The figures in the background of the plots represent the edges of the trap seen from the top and from the side.*

clearly illustrated in Figure 6-20 for a bead with density $\rho = 1050\text{kg/m}^3$ and for a flow rate of 100pL/s and a flow rate of 200pL/s. For the lower flow rate, the bead is capture by all of the traps; for the higher flow rate, the bead escapes all of the traps. For either flow rate the trajectory that the bead follows is very similar. This result seems suggests that, at least for beads starting near the trapping area, and aligned with the center of the trapping region $y = 0$, the Stokes drag force on the bead is not greatly influenced by the shape of the trap's support.

Running each of the simulations took from 20 minutes to about 2 hours and used a maximum of 400Mb of memory on an Intel Xeon 3GHz workstation with 2Gb of RAM. The simulation time was influenced by the number of collisions but also by the size of the FFT grid. For cases where the bead escaped the trap, the FFT grid had to be enlarged; enlarging the FFT grid requires recalculating the kernel transforms and makes the GMRES iterations become more expensive. The median time for generating a step candidate was 56.31s; the median time for setting up the precorrected FFT operator was 20.33s; the median time for solving (6.25) using GMRES was 37.9s.



a) Flow rate of 100pL/s.



b) Flow rate of 200pL/s.

Figure 6-20: *Trajectory of the center of mass of a bead with a diameter of 10μm and density $\rho = 1050$kg/m^3 released at $x = -5\mu$m, $y = 0\mu$m and $z = 7\mu$m. The background flow was set to that of a 3mm by 200μm channel.*

Trapping region

To further compare the trapping efficiency of the trap models and to demonstrate the use of our solver, we setup a set of simulations where a bead was released starting at $x = -10\mu$m and $z = 6\mu$ for several values of y. The density of the bead was set to $\rho = 1000$kg/m^3 and $\rho = 1050$kg/m^3 and the flow rate was set to $F = 100$pL/s, $F = 150$pL/s and $F = 200$pL/s. In this example only trap models 1 and 4 were considered. The simulations were terminated when the simulation time reached 30s or when the bead position exceeded 25μm along the x direction. The sweep consisted of 60 simulations which ran from 15 minutes to about 1.5 hours (the average run time was 45 minutes) and used a maximum of 288Mb of memory on an Intel Xeon 3GHz workstation. The sweep generated 3632 step candidates. The median time for setting up the precorrected FFT operator was 18.5s; the median time for solving (6.25) using GMRES was 29s.

The more interesting simulation results are illustrated in Figure 6-21 where it is clear that, even though the trajectory for $y = 0$ is similar for trap model 1 and trap model 4, there are some differences between the trapping behavior for “off-center” beads since, for $F = 200$pL/s, trap model 1 was able to capture some beads that trap model 4 was not able to catch.

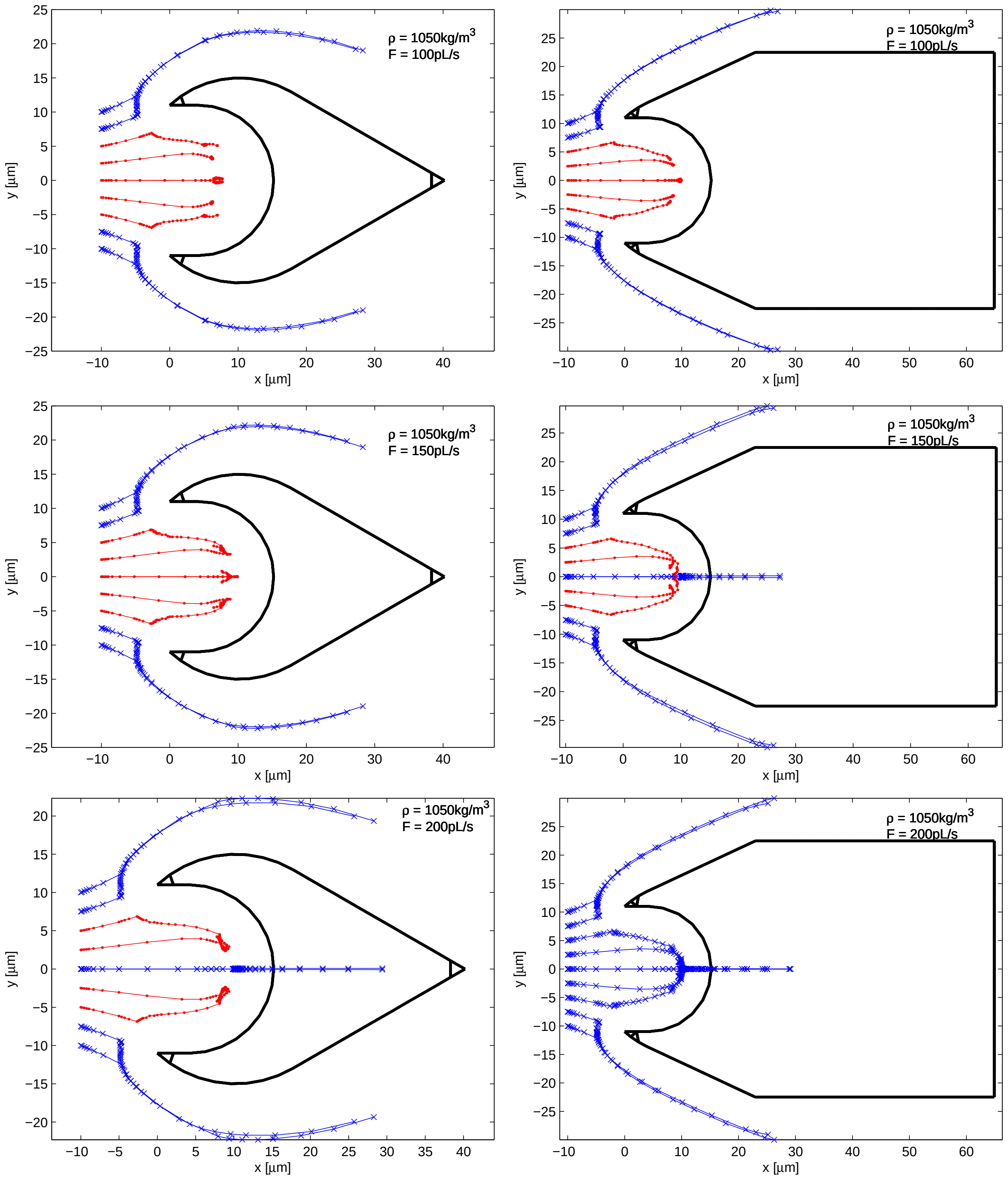


Figure 6-21: *Characterization of the trapping region for pachinko trap model 1, on the left, and pachinko trap model 4, on the right. The curves in red, with the dot marker, represent situations where the bead was trapped; the curves in blue, with the cross marker, represent situations where the bead was not captured.*

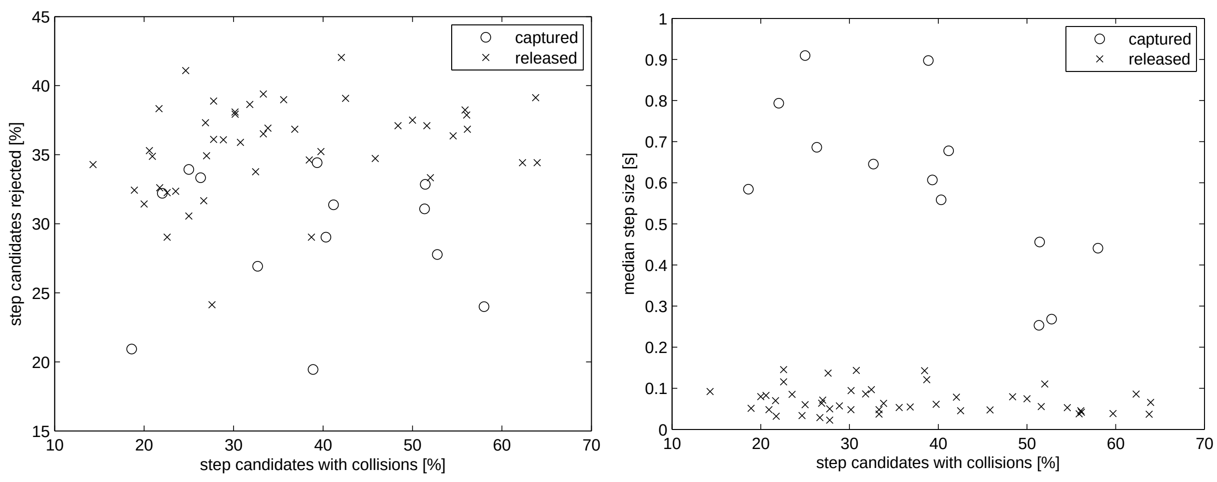


Figure 6-22: *Effect of collisions on step candidate rejection rate, on the left, and on the time integration step size, on the right. The figures illustrate compounded results from 3632 step candidates, produced by the 60 simulations that were performed for bead initial position, density and flow rate for the pachinko trap models 1 and 4.*

To illustrate the relation between the number of collisions and the step rejection ration and the step size, the percentage of the step candidates that were rejected and the step size as a function of the percentage of step candidates with collisions are illustrated in Figure 6-22. In Figure 6-22 it can be observed that for all the simulations a large percentage of the steps had collisions. It can also be observed that simulations associated with the bead being captured (and staying in a region with slower moving fluid), had less rejections and larger step sizes. The median run time for the simulations that lead to a captured bead was 1536s while the median run time for the simulations where the bead was released was 3134s.

6.8 Conclusions and future work

A stable velocity implicit time stepping scheme coupling the precorrected FFT solver presented in Chapter 4 with rigid body dynamics was introduced and demonstrated. The ODE library [27] was integrated with the solver to enable the simulation of situations involving collisions, contacts and friction. Several techniques for speeding up the calculation of each time step were presented and tested. The time integration algorithm was found to produce reasonable results.

However, it was found that some work still needs to be done to improve the robustness of the support for collisions and contacts of moving objects described by arbitrary meshes.

It was also found that, for the application examples, the distribution of panels is not very homogeneous and that, in this context, using the precorrected FFT may not be an optimal solution and that other acceleration schemes may be more advantageous. Nevertheless, the current version of the solver was still found to produce reasonable results in a reasonable amount of time and, more importantly, the velocity implicit time stepping scheme can easily be used with any other boundary element method for calculating the Stokes flow drag.

In the future, it would be worthwhile to use the updatable solver implementation to enable running parametric sweeps efficiently and to support shape optimization.

In the future it would be interesting to couple membrane models such as those described in [47], [26] and [1] with the velocity-implicit scheme and the pFFT accelerated Stokes boundary element solver.

It would be very interesting to extend this work to support slip flow boundary conditions for low Knudsen numbers (see [26]).

Appendix A - rate of energy dissipation in the fluid volume

Let S_b represent the surface of a body in an infinite quiescent fluid. Let V represent the fluid volume, bounded by the surface of the body and a surface S^{∞} at an ”infinite” distance from the object where the fluid is at rest,

$$\begin{aligned}
&\int_{S_b} u_i\sigma_{ik}n_k\mathrm{dS} + \int_{S\infty} \underbrace{u_i}_{\to 0}\sigma_{ik}n_k\mathrm{dS} = \int_V u_i\frac{\partial \sigma_{ik}}{\partial x_k} + \frac{\partial u_i}{\partial x_k}\sigma_{ik}\mathrm{dV} = \\
&\int_V \left(-\delta_{ik}\frac{\partial P}{\partial x_k} + \mu\frac{\partial}{\partial x_k}\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i}\right)\right) + \frac{\partial u_i}{\partial x_k}\left(-\delta_{ik}P + \mu\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i}\right)\right)\mathrm{dV} = \\
&\int_V \underbrace{\left(-\delta_{ik}\frac{\partial P}{\partial x_k} + \mu\frac{\partial^2 u_i}{\partial x_k\partial x_k}\right)}_{\text{=0 from Stokes equation}} + \mu\frac{\partial}{\partial x_i}\underbrace{\frac{\partial u_k}{\partial x_k}}_{=0} - P\underbrace{\frac{\partial u_k}{\partial x_k}}_{=0} + \mu\frac{\partial u_i}{\partial x_k}\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i}\right)\mathrm{dV} = \\
&\mu\int_V \frac{\partial u_i}{\partial x_k}\underbrace{\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i}\right)}_{2e_{ik}}\mathrm{dV} = 2\mu\int_V e_{ik}e_{ik}\mathrm{dV} = \mu\int_V \Phi\mathrm{dV}
\end{aligned} \tag{6.26}$$

Since $\mathbf{n}$ is the normal pointing into the body and away from the fluid, $\mathbf{f} = -\boldsymbol{\sigma}\mathbf{n}$ is the force applied to the object and the work done on the object by the Stokes drag force is

$$\int_{S_b} \mathbf{u}^T\mathbf{f}\mathrm{dS} = -\int_{S_b} u_i\sigma_{ik}n_k\mathrm{dS} = -2\mu\int_V e_{ik}e_{ik}\mathrm{dV} = -\mu\int_V \Phi\mathrm{dV} \tag{6.27}$$

which is the negative of the power dissipated in the fluid volume. In other words, the work that the body does on the fluid is dissipated in the volume.

If the body is rigid, then the velocity on S_b is given by $\mathbf{u}(\mathbf{x}) = \mathbf{v}_b + \boldsymbol{\omega}_b \times (\mathbf{x} - \mathbf{x}_b)$ and

$$\int_{S_b} \mathbf{u}^T\mathbf{f}\mathrm{dS} = \mathbf{v}_b \cdot \int_{S_b} \mathbf{f}\mathrm{dS} + \boldsymbol{\omega}_b \cdot \int_{S_b} (\mathbf{x} - \mathbf{x}_b) \times \mathbf{f}\mathrm{dS} = \mathbf{v}_b \cdot \mathbf{f} + \boldsymbol{\omega}_b \cdot \mathbf{T} = -\mu\int_V \Phi\mathrm{dV}$$

holds, which basically states that the drag force and torque on the object oppose its motion. This statement also implies that, according to the Stokes flow model, the fluid does not accumulate kinetic energy. In the Stokes flow model, the energy that a body transmits to the fluid either dissipates due to viscosity or is transmitted to other bodies in the fluid.

Chapter 7

Implementation details

In this chapter some details regarding the implementation of the precorrected FFT boundary element solver are presented.

7.1 Projection and interpolation

The projection and interpolation steps in the precorrected FFT algorithm were reviewed in Section 4.1.1. In this section we present how to actually calculate the coefficients of the projection and interpolation matrices $\mathbf{P}$ and $\mathbf{I}$.

Recall that $\mathbf{P}_{p,s} = \int_{S_s} L_{s,p}(\mathbf{x}_s)dS$ and that $\mathbf{I}_{t,i} = L_{t,i}(\mathbf{x}_t)$, for collocation testing, and $\mathbf{I}_{t,i} = \int_{S_t} L_{t,i}(\mathbf{x}_t)dS$, for Galerkin testing. Where $L_{s,p}$ is an interpolating Lagrangian polynomial on the projection stencil associated with the source panel S_s and $L_{t,i}$ is an interpolating Lagrangian polynomial on the interpolation stencil associated with the test panel S_t.

For practical reasons $\mathbf{P}_{t,i} = L_{t,i}(\mathbf{x}_t)$ is calculated in two steps. In the first step, a set of monomials of the $\mathbf{x}_t$ minus the center of the interpolation stencil associated with the test panel t are calculated. In the second step, the values of the monomials are combined to produce $L_{t,i}(\mathbf{x}_t)$. Note that the monomial coefficients for the interpolating polynomials, which are used to combine the monomials into $L_{t,i}(\mathbf{x}_t)$, do *not* depend on $\mathbf{x}_t$ and are a function of only the interpolation stencil and the grid spacing. If the same type of interpolation stencil is used for all the targets, these coefficients can be computed once and then reused for the calculation of all the $L_{t,i}(\mathbf{x}_t)$ entries. If Galerkin testing is being used, $\mathbf{P}_{t,i} = \int_{S_t} L_{t,i}(\mathbf{x}_t)dS$ and, in the second step, rather than combining monomial function values, the algorithm uses the same method to combine the appropriate moments over S_t (note that these moments are centered on the interpolation stencil associated with the test panel t). The same process is used to calculate $\mathbf{P}_{p,s}$.

The monomial coefficients $\mathbf{c}^{*,i}$ for the interpolating polynomial $L_{*,i}$ can be calculated by solving, in the least squares sense, a linear system $\mathbf{A}\mathbf{c}^{*,i} = \mathbf{e}^i$, where

$$A_{j,k} = (x_j - x_0)^{m_k}(y_j - y_0)^{n_k}(z_j - z_0)^{p_k} \tag{7.1}$$

where $\mathbf{x}_j$ is the jth stencil point and $\mathbf{x}_0$ is the stencil center; and $\mathbf{e}^i_j = \delta_{i,j}$. Since (7.1) is a Vandermonde-like matrix its condition number will deteriorate rapidly with m, n and p. Moreover, if the entries in $\mathbf{A}$ are not rescaled, solving the linear system (7.1), even using SVD based methods, will not yield accurate values for $\mathbf{c}^{*,i}$ and will instead produce a low order approximating polynomial that will significantly compromise the accuracy of the pFFT algorithm. Because it does not lead to an immediate catastrophic failure of the pFFT algorithm, the loss of accuracy due to failing to rescale $\mathbf{A}$, can be a very hard to find bug.

A version of our implementation of the projection/interpolation algorithm is included in Listing 7.1 and Listing 7.2.

Listing 7.1: *Functor used for projection and interpolation using monomial basis functions and panel moments.*

```
template <class Grid, class operation_tag>
struct project_on_grid_functor_impl
{
  typedef geometric_element::moment_container moment_container ;

  template <class Stencil>
  project_on_grid_functor_impl(Grid const & grid, Stencil const & stencil)
    : _grid(grid), _moments(new moment_container())
  {
    position center_point = grid.position_from_grid_coordinates(stencil.center()) ;
    vector<position> stamp_point_positions ;
    for (typename Stencil::stamp_iterator stamp_it = stencil.stamp_begin() ;
       stamp_it != stencil.stamp_end() ; ++stamp_it)
    {
      vector<int> stamp_coordinates = *stamp_it ;
      for (unsigned dim = 0 ; dim != 3 ; ++dim)
```

```
        stamp_coordinates[dim] += -stencil.bounds()[dim].first ;

      stamp_point_positions.push_back(
        grid.position_from_grid_coordinates(stamp_coordinates) - center_point) ;
    }

    // Calculate monomial values on the stencil points
    // establishing a basis_values_on_stencil matrix.
    f_monomials interpolating_function(
      stencil.span()[0], stencil.span()[1], stencil.span()[2]) ;

    ublas::matrix<double, ublas::column_major> basis_values_on_stencil =
      detail::evaluate_interpolation_functions_on_stencil(
        stamp_point_positions, interpolating_function) ;

    // SCALE the rows of the basis_values_on_stencil matrix
    vector<double> scaling(basis_values_on_stencil.size1(), 0) ;
    for (unsigned row = 0 ; row != basis_values_on_stencil.size1() ; ++row)
    {
      // Determine the maximum value on the column
      for (unsigned col = 0 ; col != basis_values_on_stencil.size2() ; ++col)
        scaling[row] = max(fabs(basis_values_on_stencil(row, col)),
          scaling[row]) ;

      // Scale the columns in this row
      for (unsigned col = 0 ; col != basis_values_on_stencil.size2() ; ++col)
        basis_values_on_stencil(row, col) /= scaling[row] ;
    }

    // Calculate the pseudo inverse of the scaled matrix
    interpolating_polynomial_coefficients_m =
      pseudo_inverse(basis_values_on_stencil) ;

    // Scale the columns of the pseudo inverse
    for (unsigned col = 0 ; col != interpolating_polynomial_coefficients_m.size2() ; ++col)
    for (unsigned row = 0 ; row != interpolating_polynomial_coefficients_m.size1() ; ++row)
      interpolating_polynomial_coefficients_m(row, col) /= scaling[col] ;

    // Resize the storage for interpolating_polynomial_coefficients
    ...

    // Reorganize the the interpolating_polynomial_coefficients
    // to facilitate the computation of the projection and interpolation
    // coefficients.
    _moment_orders = interpolating_function.term_orders() ;
    unsigned n_functions = interpolating_function.size() ;
    for (unsigned fi = 0 ; fi != n_functions ; ++fi)
    {
      unsigned const ix = _moment_orders[fi][0] ;
      unsigned const iy = _moment_orders[fi][1] ;
      unsigned const iz = _moment_orders[fi][2] ;

      for (unsigned pi = 0 ; pi != stencil.size() ; ++pi)
        interpolating_polynomial_coefficients[pi][ix][iy][iz] =
          interpolating_polynomial_coefficients_m(pi, fi) ;
    }

    // Determine the maximum order of the moments that will be required
    // to project the sources onto the projection grid using the stencil.
    _max_moment_order = 0 ;
    for (unsigned io = 0 ; io != _moment_orders.size() ; ++io)
      _max_moment_order = max(_max_moment_order,
      _moment_orders[io][0] + _moment_orders[io][1] + _moment_orders[io][2]) ;

    // Determine the minimum and maximum positions for the stencil
    // centers (in grid coordinates).
    for (unsigned dim = 0 ; dim != 3 ; ++dim) {
      _min_grid_coordinates.push_back(-stencil.bounds()[dim].first) ;
      _max_grid_coordinates.push_back((int(_grid.num_points()[dim]) - 1)
      - stencil.bounds()[dim].second) ;
    }
  }

  // Calculate the projection or interpolation coefficients for
  // a given source or target.
  // The function input is a tuple taking
  // (geometric_element const * source/target,
  //  grid_coordinate const & nearest_grid_coordinate,
  //  projection_coefficient & output).
  template <class Tuple>
  void operator() (Tuple const & t) const
  {
    // Extract the tuple components.
    geometric_element const * e = boost::get<0>(t) ;
    using namespace boost::tuples ;

    typedef typename element<1,Tuple>::type grid_coordinate_ref ;
    grid_coordinate_ref nearest_grid_coordinates = boost::get<1>(t) ;

    typedef typename element<2,Tuple>::type projection_coefficients_ref ;
    projection_coefficients_ref projection_coefficients = boost::get<2>(t) ;

    // If the nearest grid point is too close to the grid border move it
    // in such that the projection stencil points are contained in the grid.
    position centroid = e->get_centroid() ;
    _grid.grid_coordinates_from_position(centroid, nearest_grid_coordinates.begin()) ;
    for (unsigned dim = 0 ; dim != 3 ; ++dim)
      nearest_grid_coordinates[dim] = bound_value(
      _min_grid_coordinates[dim], _max_grid_coordinates[dim],
      nearest_grid_coordinates[dim]) ;

    // Determine the location of the projection center grid point in world coordinates.
    position projection_center = _grid.position_from_grid_coordinates(
      nearest_grid_coordinates) ;

    // Make sure the moment result container has enough space for the moments we need
    // for this source or target type.
    typedef typename boost::remove_reference<projection_coefficients_ref>::type
      projection_coefficient_container ;
    typedef typename projection_coefficient_container::value_type
      projection_coefficient_type ;
    unsigned const extra_moment_order =
    combine_moments_and_stencil_interpolation_coefficients<
      projection_coefficient_type>::extra_required_moment_order ;

    resize_panel_moment_container(extra_moment_order) ;

    // Use the position of the nearest projection grid point as the center for
    // calculating the moments of the source.
    e->get_moments(_max_moment_order + extra_moment_order, projection_center, *_moments) ;

    // Fill in the projection coefficients
    for (unsigned pi = 0 ; pi != interpolating_polynomial_coefficients.size() ; ++pi)
      combine_moments_and_stencil_interpolation_coefficients<
        projection_coefficient_type>::apply(
          _moment_orders, interpolating_polynomial_coefficients[pi],
          e, centroid, *_moments, projection_center, projection_coefficients[pi]) ;
  }
  ...
} ;
```

Listing 7.2: *Function that combines the coefficients for the stencil interpolation polynomials with the moments evaluated for a given source or target panel to calculate the projection or interpolation coefficients.*

```
template <>
struct combine_moments_and_stencil_interpolation_coefficients<
  ...::projection_coefficient_type>
{
  enum { extra_required_moment_order = 1 }  ;

  template <...>
  inline static void apply(
    MomentOrders const & moment_orders,
    StencilInterpolationCoefficients const & stencil_interpolation_coefficients,
    GeometricElement const * element,
    Centroid const & centroid,
    Moments const & moments,
    ProjectionCenter const & projection_center,
    Coefficient & result)
  {
    result.clear() ;

    // The extra zt or zs moment order term is in absolute
    // coordinates so we need to add the base term to the
    // local z^(p+1) term i.e.
    // x^m*y^n*z^p*(Z + z) <-- x^m*y^n*z^(p+1) + Z*x^m*y^n*z^p.
    for (unsigned io = 0 ; io != moment_orders.size() ; ++io)
    {
      unsigned const ox = moment_orders[io][0] ;
      unsigned const oy = moment_orders[io][1] ;
      unsigned const oz = moment_orders[io][2] ;
      result[0] += stencil_interpolation_coefficients[ox][oy][oz]
        * moments[ox][oy][oz] ;
      result[1] += stencil_interpolation_coefficients[ox][oy][oz]
        * (moments[ox][oy][oz + 1] + moments[ox][oy][oz] * projection_center[2]) ;
    }
  }
};
```

7.2 Exploiting kernel symmetries to reduce memory usage

The Stokes flow Green's functions, as well as many other Green's functions for physical problems, have symmetries. If the Green's function has symmetries and is evaluated on a regular grid properly aligned with its axis and center of symmetry, its discrete Fourier transform (DFT) also has the same symmetries and the storage for the DFT can be *compressed* accordingly. Moreover, if the Green's function is real and it is either symmetric or antisymmetric along each axis, its transform is either purely real or purely imaginary and its DFT can be stored using a collection of real values rather than a collection of complex values.

The precorrected FFT algorithm computes the DFT of the Green's functions to accelerate the calculation of the convolution of the Green's function and the projected forces on a regular grid. If the Green's function or the projected forces are real, conjugate symmetry can also be used to reduce storage. In our implementation the storage for the DFT of real signals is *compressed* by truncating the DFT such that it has $\lfloor N_z/2 \rfloor + 1$ entries along the "last" dimension, which we assume to be the z direction for a 3D grid (the truncation could have been performed along any other axis direction but we use FFTW's [54] `r2c` transforms and thus follow their convention).

The velocities on the regular grid are computed by zero padding the projected forces, calculating the DFT of the zero padded forces on the grid; calculating the pointwise multiplication between the DFT of the zero padded forces on the grid and the DFT of the Green's function; and then inverse transforming the result of the pointwise multiplication and removing the padding.

The complicated part of the FFT accelerated convolution is the calculation of the pointwise multiplication between the *compressed* DFT of the Green's function and the *compressed* DFT of the projected forces, where it is assumed that the projected forces do not have any symmetries that can be exploited. For that purpose, we present an algorithm in Listing 7.3 that can compute the pointwise multiplication of the DFT of an input signal with the DFT of a Green's function with any combination of odd symmetry, even symmetry or asymmetry along any of the axis function and that works for any number of dimensions.

Listing 7.3: *Function that calculates the pointwise product of a kernel, with symmetries, and a signal and accumulates the result onto a second signal.*

```
#include "util/get_negated_functor.hpp"

// nd_multiply_accumulate_transform_with_symmetries
template <
  class a_it_type,
  class a_size_it_type,
  class a_stride_it_type,
  class a_symmetry_it_type,
  class b_it_type,
  class b_size_it_type,
  class b_stride_it_type,
  class out_it_type,
  class out_size_it_type,
  class out_stride_it_type,
  class F>
inline void nd_multiply_accumulate_transform_with_symmetries_impl(
  size_t            rank,
  a_it_type              a_it,
  a_size_it_type         a_size_it,
  a_stride_it_type       a_stride_it,
  a_symmetry_it_type     a_symmetry_it,
  b_it_type              b_it,
  b_size_it_type         b_size_it,
  b_stride_it_type       b_stride_it,
  bool                   signal_is_real,
  out_it_type            out_it,
  out_size_it_type       out_size_it,
  out_stride_it_type     out_stride_it,
  F const & f)
{
  a_it_type const a_begin = a_it ;
  ignore_unused_variable_warning(a_begin) ;

  size_t const a_stride = *a_stride_it ;
  size_t const b_stride = *b_stride_it ;
  size_t const out_stride = *out_stride_it ;

  size_t const a_size = *a_size_it ;
  size_t const b_size = *b_size_it ;
  size_t const out_size = *out_size_it ;

  if (rank != 1 && *a_symmetry_it == odd) assert(a_size == (b_size - 1)/2) ;
  if (rank != 1 && *a_symmetry_it == even) assert(a_size == b_size / 2 + 1) ;
  assert(out_size == b_size) ;

  a_it_type const a_end = a_it + a_size * a_stride ;
  b_it_type const b_end = b_it + b_size * b_stride ;
  out_it_type const out_end = out_it + out_size * out_stride ;
  ignore_unused_variable_warning(out_end) ;

  // If the kernel is odd it does not store the first value,
  // which is zero.
  if (*a_symmetry_it == odd) {
    b_it += b_stride ;
    out_it += out_stride ;
  }

  if (rank > 1) {
    while (a_it != a_end) {
      nd_multiply_accumulate_transform_with_symmetries_impl(
        rank - 1,
        a_it, a_size_it + 1, a_stride_it + 1, a_symmetry_it + 1,
        b_it, b_size_it + 1, b_stride_it + 1, signal_is_real,
        out_it, out_size_it + 1, out_stride_it + 1,
        f) ;

      a_it += a_stride ;
      b_it += b_stride ;
      out_it += out_stride ;
    }

    // Get a_it back in the data range
    a_it -= a_stride ;

    if (b_size % 2 == 0) {
      // If using symmetry and signal size is even, point a to
      // the second to last entry.
      if (*a_symmetry_it == even) {
        // signal is A B C D E F
        // kernel is a b c d c b
        // processing E next so must point kernel to c
        a_it -= a_stride ;
      }
      else if (*a_symmetry_it == odd) {
        // signal is A B C D E F
        // kernel is 0 b c 0 -c -b
        // precessing D now so just skip to E
        b_it += b_stride ;
        out_it += out_stride ;
      }
    }
    else {
      // Nothing to do because:

      // if kernel is even:
      // signal is A B C D E
      // kernel is a b c c b
      // b_it should point to D and a_it to c already

      // if kernel is odd:
      // signal is A B C D E
      // kernel is a b c -c -b
      // b_it should point to D and a_it to c already

      // is kernel is asymmetric:
      // signal is A B C D E
      // kernel is a b c d e
      // b_it should be b_end
    }

    if (b_it == b_end)
       return ;

    if (*a_symmetry_it == even)  {
      // Now go back with the same f
      while (b_it != b_end) {
        nd_multiply_accumulate_transform_with_symmetries_impl(
          rank - 1,
          a_it, a_size_it + 1, a_stride_it + 1, a_symmetry_it + 1,
          b_it, b_size_it + 1, b_stride_it + 1, signal_is_real,
          out_it, out_size_it + 1, out_stride_it + 1,
          f) ;

        a_it -= a_stride ;
        b_it += b_stride ;
        out_it += out_stride ;
      }
    }
    else if (*a_symmetry_it == odd) {
      typename get_negated_functor_impl<F>::type neg_f(
        get_negated_functor(f)) ;

      // Now go back with a negated f
      while (b_it != b_end) {
        nd_multiply_accumulate_transform_with_symmetries_impl(
          rank - 1,
          a_it, a_size_it + 1, a_stride_it + 1, a_symmetry_it + 1,
          b_it, b_size_it + 1, b_stride_it + 1, signal_is_real,
          out_it, out_size_it + 1, out_stride_it + 1,
          neg_f) ;

        a_it -= a_stride ;
        b_it += b_stride ;
        out_it += out_stride ;
      }
    }
  }
  else // rank == 1
  {
    // This branch has the same structure as the branch for rank > 1
    // except that instead of recursively calling this function
    // the code calls the functor f or neg_f instead.
    while (a_it != a_end) {
      f(*out_it, *a_it, *b_it) ;
      a_it += a_stride ;
      b_it += b_stride ;
      out_it += out_stride ;
    }

    // Get a_it back in the data range
    a_it -= a_stride ;

    if (signal_is_real || (b_size % 2 == 0)) {
      // If using symmetry and signal size is even, point a to
      // the second to last entry.
      if (*a_symmetry_it == even) {
        // signal is A B C D E F
        // kernel is a b c d c b
        // processing E next so must point kernel to c
        a_it -= a_stride ;
      }
      else if (*a_symmetry_it == odd) {
        // signal is A B C D E F
        // kernel is 0 b c 0 -c -b
        // precessing D now so just skip to E
        b_it += b_stride ;
        out_it += out_stride ;
      }
    }
    else {
      // Nothing to do for same reasons as in corresponding
      // case in the rank > 1 branch above.
    }

    if (b_it == b_end)
    return ;

    if (*a_symmetry_it == even) {
      while (b_it != b_end) {
        f(*out_it, *a_it, *b_it) ;
        a_it -= a_stride ;
        b_it += b_stride ;
        out_it += out_stride ;
      }
    }
    else if (*a_symmetry_it == odd) {
      typename get_negated_functor_impl<F>::type neg_f(
        get_negated_functor(f)) ;

      // Now go back with a flipped f
      while (b_it != b_end) {
        neg_f(*out_it, *a_it, *b_it) ;
        a_it -= a_stride ;
        b_it += b_stride ;
        out_it += out_stride ;
      }
    }
  }
}
```

Note that the algorithm presented in Listing 7.3 takes a functor `f`. The functor `f` is used, together with `get_negated_functor` to enable the flexible and efficient configuration of the function to be performed. For example, the functor `f` can be a `multiply_accumulate` functor, coupled to a `multiply_subtract` functor by `get_negated_functor`. Using functors to represent the fundamental multiply accumulate operation does not incur in any performance penalty whereas using a factor of 1 or -1 to multiply the Green's function would be less general and would introduce an unnecessary multiplication. It would have been possible to avoid using functors and to use expression templates [28, 29] instead to represent the multiply accumulate and multiply subtract as `*out_it += *a_it **b_it` and `*out_it -= *a_it * *b_it` without incurring in performance loss but we opted for using functors and `get_negated_functor` because it was simpler.

To illustrate the use of `nd_multiply_accumulate_transform_with_symmetries_impl`, a driver routine is presented in Listing 7.4 where the appropriate instance of the function is called depending on the symmetries of the Green's function and on the signal being real or not. When the Green's function is symmetric or antisymmetric along all the directions and it is stored as a set of real values, depending on the number of directions where the function is anti-symmetric, the transform values will be multiplied by -1 and/or by i. The multiplication by -1 is achieved at no cost by using `multiply_subtract` instead of `multiply_accumulate`. The multiplication by i is achieved at no cost by adapting the iterator over the transform data such that a special type is returned when the iterator is dereferenced. Specializations of the multiplication operator for the purely imaginary type are defined and inlined such that using the `imaginary_iterator_adaptor` does not result in a performance penalty.

Listing 7.4: *Function that calculates the pointwise product of a kernel, with symmetries, and a signal and accumulates the result on a second signal*

```
// Driver routine for pointwise kernel signal multiply accumulate
// operation.
template <
  class output_scalar_type,
  class kernel_scalar_type,
  class input_scalar_type,
  class F>
inline void nd_multiply_accumulate_transform_with_symmetries(
  nd_signal_transform<output_scalar_type> & accumulator,
  nd_kernel_transform<kernel_scalar_type> const & kernel,
  nd_signal_transform<input_scalar_type>  const & signal,
  F const & f)
{
  complex<int> symmetry_factor(1, 0) ;

  vector<function_symmetry_type> const & kernel_symmetry = kernel.symmetry() ;
  size_t const n_dims = kernel_symmetry.size() ;

  for (size_t dim = 0 ; dim != n_dims ; ++dim)
    if (kernel_symmetry[dim] == odd)
      symmetry_factor *= complex<int>(0, -1) ;

   bool const negate = (symmetry_factor.real() == -1)
     || (symmetry_factor.imag() == -1) ;
   bool const use_imaginary_iterator_adaptor = kernel.transform_data_is_real()
     && (symmetry_factor.real() == 0) ;

  if (kernel.transform_data_is_real()) {
    if (negate) {
      if (use_imaginary_iterator_adaptor) {
        nd_multiply_accumulate_transform_with_symmetries(
          imaginary_iterator_adaptor(kernel.transform_data_as_real()),
          begin(kernel.transform_size()),
          begin(kernel.transform_stride()),
          begin(kernel.symmetry()),
          signal.transform_data_as_complex(),
          begin(signal.transform_size()),
          begin(signal.transform_stride()),
          nd_signal_transform<input_scalar_type>::signal_is_real,
          accumulator.transform_data_as_complex(),
          begin(accumulator.transform_size()),
          begin(accumulator.transform_stride()),
          get_negated_functor(f)) ;
      }
      else {
        nd_multiply_accumulate_transform_with_symmetries(
          kernel.transform_data_as_real(),
          ... // Commented out repeated parameters
          get_negated_functor(f)) ;
      }
    }
    else {
      if (use_imaginary_iterator_adaptor) {
        nd_multiply_accumulate_transform_with_symmetries(
          imaginary_iterator_adaptor(kernel.transform_data_as_real()),
          ... // Commented out repeated parameters
          f) ;
      }
      else {
        nd_multiply_accumulate_transform_with_symmetries(
        kernel.transform_data_as_real(),
        ... // Commented out repeated parameters
        f) ;
      }
    }
  }
  else {
    if (negate) {
      nd_multiply_accumulate_transform_with_symmetries(
        kernel.transform_data_as_complex(),
        ... // Commented out repeated parameters
        get_negated_functor(f)) ;
    }
    else {
      nd_multiply_accumulate_transform_with_symmetries(
        kernel.transform_data_as_complex(),
        ... // Commented out repeated parameters
        f) ;
    }
  }
}
```

Please note that, if the input signal is real and the kernel is complex, or vice versa, then the output signal will be complex and the pointwise multiplication algorithm will only work if the uncompressed transform for the input signal is provided. It is likely that this restriction can be lifted but it was decided to leave that improvement as future work.

In our implementation, compressing the DFT for the Green's function is left to FFTW's `r2c` or `r2r` routines depending on the Green's function symmetries. Using the `r2r` transforms has the advantage that the Green's functions only need to be evaluated on a quadrant of the domain. However, the `r2r` transforms are only applicable when the function is symmetric or antisymmetric along all the directions and will not work for the image Green's functions because they are not sampled around their center of symmetry. It would have been possible, and more general, to just use FFTW's `r2c` or `c2c` transforms and to compress the resulting DFT afterwards. In hindsight, doing so, or using a `r2r` first along the symmetric directions and then using `r2c` along the non-symmetric directions would have been better than just relying on `r2r` to compress storage. Nevertheless, modifying the implementation is trivial and it is not the focus of this section. Moreover, for planar topologies, the layered transforms, presented in Section 7.3 do exploit all possible symmetries for both image and non-image Green's functions.

7.3 Specializations for planar topologies

Especially for surface micromachined devices, the problem dimensions along the z direction, normal to the substrate, are usually much smaller than the dimensions along the x and y directions, parallel to the substrate. For these problems, the number of FFT grid points along the vertical direction N_z is much smaller than the number of grid points along the x and y directions. It turns out that, for small enough N_z, it is faster and more memory efficient to compute the grid convolution by layers along the z direction and using 2D accelerated convolution for each interacting layer pair, than using a full 3D FFT accelerated convolution.

In the layered convolution method the input signal and the Green's function on the grid are Fourier transformed along the x and y direction but not along the z direction. This has the immediate advantage that it removes the need for padding the input and output signals along the z direction. On the other hand the layered convolution requires N_z^2 layer to layer 2D convolutions, as can be observed in Listing 7.5.

A major advantage for the layered convolution method when applied to problems involving image Green's functions is that, the symmetry along the x and y axis can be fully exploited and the transform data can be stored a real vector rather than a vector of complex values. Meanwhile, for the 3D FFT convolution method the image Green's functions are evaluated in a way that symmetry along the z direction cannot be used and so the transform data cannot be stored as a real vector or computed directly using FFTW's `r2r` routines.

Another advantage of the layered convolution method is that calculating the transform of the projected image panel source panel distribution on the grid from the transform of the projected source panel distribution on the grid only requires a simple re-indexing operation.

Listing 7.5: *Layered convolution.*

```
template <class scalar_type>
void layered_kernel_transform<scalar_type>::convolve_accumulate(
    input_signal<scalar_type> const & from,
    output_signal<scalar_type> & accumulator,
    scalar_type const & factor) const
{
  if (factor == scalar_type(0)) return ;

  layered_signal_transform<scalar_type> const * from_p =
    boost::polymorphic_downcast<layered_signal_transform<scalar_type> const *>(&from) ;
  layered_signal_transform<scalar_type> * to_p =
    boost::polymorphic_downcast<layered_signal_transform<scalar_type>*>(&accumulator) ;

  // Explicitly perform an N^2 convolution along the layering direction
  int const min_from_layer = from_p->min_layer_index() ;
  int const max_from_layer = from_p->max_layer_index() ;
  int const min_to_layer = to_p->min_layer_index() ;
  int const max_to_layer = to_p->max_layer_index() ;

  for (int from_layer = min_from_layer ; from_layer <= max_from_layer ; ++from_layer)
    for (int to_layer = min_to_layer ; to_layer <= max_to_layer ; ++to_layer)
      layer(to_layer - from_layer).convolve_accumulate(
        from_p->layer(from_layer), to_p->layer(to_layer), factor) ;
}
```

Note that the convolution interface provides a virtual mechanism to perform the convolution which hides the actual implementation. In other words, the user of the convolution classes does not need to be aware of whether a layered convolution or a full 3D FFT accelerated convolution is being used. The convolution interface also takes care of any necessary padding and unpadding.

7.4 Precorrection

The precorrection algorithm is responsible for subtracting the grid based interactions between nearby source-target pairs. However, a naive implementation of the precorrection algorithm can be very inefficient. In this section the techniques used in our implementation of the precorrection algorithm are presented. To facilitate the discussion we first introduce some of the variables and types that will appear below:

- `interaction_values` is a sparse matrix with entries of type `interaction_value-_type`. This matrix has `num_sources` columns and `num_targets` rows.
- `interaction_list` is a collection of `num_sources` collections of target indices and is basically a representation of the sparse structure of `interaction_values`.
- `projection_coefficients` represent the projection weights that map the source forces to forces and force moments on the grid. This is a collection of collections of `projection_coefficient_type`
- `projection_grid_coordinates` represent grid coordinates used to map the source forces to forces and force moments on the grid.
- `interpolation_coefficients` represents the interpolation weights used to map the grid velocities and velocity moments to velocities on the target evaluation points, if collocation testing is being used, or integrals of the velocity over the target panels, if Galerkin testing is being used. This variable is a collection of collections of elements of type `interpolation_coefficient_type`.
- `interpolation_grid_coordinates` represent the grid coordinates used to map the grid velocities and velocity moments to velocities on the target.
- `greens_function(t,s)` is a function that takes a target point `t` and a source point `s` and returns the value of the Green's function of type `kernel_value_type`.
- `projected_kernel_value_type` is the result of the product between a projection coefficient and a kernel value.
- `multiply_accumulate(pg,p,g)` multiplies `p` of type `projection_coefficient-_type` and `g` of type `kernel_value_type` and accumulates the result onto an element of the type `projected_kernel_value_type`.
- `multiply_subtract(iv,i,pg)` multiplies `i` of type `interpolation_coefficient-_type` by `pg` of type `projected_kernel_value_type` and subtracts the result from `iv` of type `interaction_value_type`.

A straightforward, but inefficient, implementation of the precorrection algorithm is presented in Listing 7.6 where, for convenience, it is assumed that the `*` operator works in a manner that is consistent with the `multiply_accumulate` operation between elements of `projection_coefficient_type` and of `kernel_value_type`.

Listing 7.6: *Basic precorrection algorithm.*

```
for si = 1 : num_sources,
   for ti = interaction_list{si}
      for pci = 1 : length(projection_coefficients(:,si)),
         for ici = 1 : length(interpolation_coefficients(:,ti)),
            multiply_subtract(interaction_values(ti,si), ...
               interpolation_coefficients(ici, ti), ...
               projection_coefficients(pci, si) * ...
               greens_function(...
                  position(interpolation_grid_coordinates(:,ici,ti)),...
                  position(projection_grid_coordinates(:,si,ti)))
         end
      end
   end
end
```

There are several sources of inefficiency in this implementation. The more important deficiency of the algorithm is that the work done to calculate the kernel projected at a given point is repeated for multiple panels in `interaction_list(si)`.

The first source of inefficiency can be addressed by separating the precorrection for each source into a *scatter* phase and a *gather* phase. In the *scatter* phase, the list of the interpolation points associated with the targets in the source's interaction list is computed. Then the sum of the projected kernel values due to the source's projection grid points is calculated at the interpolation points in the list above. The contributions from all the projection points at each interpolation point are accumulated and stored in elements of the type `projected_kernel_value_type`. In the *gather* phase, for each target in the source's interaction list, the projected kernel values corresponding to the interpolation points associated with that target are multiplied by the corresponding interpolation coefficients and subtracted from the appropriate entry in `interaction_values` using the operation `multiply_subtract`.

Another source of inefficiency is that `greens_function(t,s)` is evaluated multiple times for the same `(t,s)` pair and that the positions `t` and `s` are also being repeatedly recalculated. Dealing with this issue is important if the cost of evaluating `greens_function` and computing positions from grid coordinates is high. Unfortunately, caching `greens_function(t,s)` as a function of both `t` and `s` would either require a large amount of memory or would would use some sort of associative container with a non-trivial access time. However, if `greens_function(t,s)` is translation invariant i.e. if `greens_function(t,s)` is the same as `greens_function(t-s,0)` the values of `greens_function` can be reused and accessed as a function of `t-s` without requiring an excessive amounts of storage.

In our implementation, presented in Listing 7.7 the following approach for caching and accessing the values of the Green's function was used: First, the maximum grid-based distance between a projection grid coordinate of a source and an interpolation grid coordinate of an interacting target was calculated. Calculating the maximum grid-based interaction range benefits significantly from first computing a grid-aligned bounding box for the projection grid coordinates of each source and for the interpolation grid coordinates of each target. Once the maximum interaction range is computed the values of `greens_function(t-s,0)` in that range are calculated and stored in and array `kernel-_values` of `kernel_value_type` elements. Let $R_{min,k}$ and $R_{max,k}$ represent the minimum and maximum values of $t - s$ along direction k, the span of the interactions along that direction is $S_k = R_{max,k} - R_{min,k} + 1$. Let T_k be the stride for `kernel_values`, defined in a manner consistent with S_k. For a source grid coordinate `s` and a target grid coordinate `t`, the linear index into `kernel_values` is given by

$$\texttt{cached\_kernel\_index} = \sum_k (t_k - s_k - R_{\mathrm{min,k}})T_k. \tag{7.2}$$

Instead of computing (7.2) for each access to `kernel_values`, linear indices `projection-_green_offset` = $\sum_k (s_k + R_{\mathrm{min,k}})T_k$, for each projection point, and `interpolation_green-_offset` = $\sum_k t_k T_k$, for each interpolation point, are computed and saved once. The difference between the two linear indices is then used to access the cached kernel values.

Listing 7.7: *Precorrection algorithm.*

```
// project_on_grid performs the scatter operation for each
// source. See subtract_grid_based_interactions below for
// usage details.
template <...>
void project_on_grid(
  target_grid_it_type                target_grid_it,
  target_grid_it_type               target_grid_it_end,
  target_green_it_type              target_green_it,
  projection_green_offset_it_type   projection_green_offsets_begin,
  projection_green_offset_it_type   projection_green_offsets_end,
  projection_coefficients_it_type   projection_coefficients_begin,
  kernel_container_type const &     kernel_values,
  projected_kernel_container_type & projected_kernel_values)
{
  projected_kernel_value_type v ;

  // Calculate the projection of the source onto each target point.
  while (target_grid_it != target_grid_it_end)
  {
    size_t const target_grid_offset = *target_grid_it ;
    int const target_green_offset = *target_green_it ;

    // Zero out accumulator.
    clear(v) ;

    projection_green_offset_it_type projection_green_offset_it
      = projection_green_offsets_begin ;
    projection_coefficients_it_type projection_coefficients_it
      = projection_coefficients_begin ;
    while (projection_green_offset_it != projection_green_offsets_end)
    {
      int const cached_kernel_index = target_green_offset
        - *projection_green_offset_it ;

      multiply_accumulate(
        v, kernel_values[cached_kernel_index],
        *projection_coefficients_it) ;

      ++projection_coefficients_it ;
      ++projection_green_offset_it ;
    }

    projected_kernel_values[target_grid_offset] = v ;

    ++target_grid_it ;
    ++target_green_it ;
  }
}

template <
  class kernel_value_type,
  class projected_kernel_value_type,
  class interaction_value_type,
  ...>
void subtract_grid_based_interactions(
  KernelEvaluationFunctor const &           kernel_evaluation_functor,
  InteractionList const &                   interaction_list,
  ProjectionGridCoordinates const &         projection_grid_coordinates,
  ProjectionOffsets const &                 projection_offsets,
  ProjectionCoefficients const &            projection_coefficients,
  InterpolationGridCoordinates const &      interpolation_grid_coordinates,
  InterpolationOffsets const &              interpolation_offsets,
  InterpolationCoefficients const &         interpolation_coefficients,
  InteractionValuesIterator const &         interaction_values)
{
  // Determine the maximum interaction range.
  vector<pair<int, int> > interaction_range =
  calculate_maximum_grid_based_interaction_range(
  interaction_list,
  projection_grid_coordinates,
  interpolation_grid_coordinates) ;

  // Setup a mini-grid from the interaction range. */
  size_t const n_dims = interaction_range.size() ;
```

```
vector<vector<int> > kernel_grid_indices ;
vector<int> green_min_coordinates, green_max_coordinates ;
size_t num_green_grid_points = 1 ;
for (size_t dim = 0 ; dim != n_dims ; ++dim) {
  kernel_grid_indices.push_back(
    linspace(interaction_range[dim].first,
    1, interaction_range[dim].second)) ;

  green_min_coordinates.push_back(interaction_range[dim].first) ;
  green_max_coordinates.push_back(interaction_range[dim].second) ;
  num_green_grid_points *= kernel_grid_indices.back().size() ;
}

// Calculate kernel values
vector<kernel_value_type> kernel_values(num_green_grid_points) ;
vector<size_t> green_stride = kernel_evaluation_functor(
kernel_grid_indices, kernel_values.begin()) ;

int green_origin_offset = -linear_index_from_stride_and_multi_index(
green_stride, green_min_coordinates) ;

size_t const n_sources = interaction_list.size() ;
vector<int> interpolation_green_offsets ;
vector<int> projection_green_offsets ;

// Allocate workspace to contain the projected kernel values.
using max ;
size_t const n_targets = boost::size(interpolation_offsets) ;
size_t max_interpolation_offset = 0 ;
for (size_t ti = 0 ; ti != n_targets ; ++ti)
  for (size_t poi = 0 ; poi != interpolation_offsets[ti].size() ; ++poi)
    max_interpolation_offset = max(
      interpolation_offsets[ti][poi], max_interpolation_offset) ;

size_t max_projection_offset = 0 ;
  for (size_t si = 0 ; si != n_sources ; ++si)
    for (size_t poi = 0 ; poi != projection_offsets[si].size() ; ++poi)
      max_projection_offset = max(
        projection_offsets[si][poi], max_projection_offset) ;

vector<projected_kernel_value_type> projected_kernel_values(
  max(max_interpolation_offset, max_projection_offset) + 1);

// Map grid offsets into (aliased) offsets into the kernel mini-grid.
vector<int> green_offset_from_grid_offset(max_interpolation_offset + 1, 0) ;
  for (size_t ti = 0 ; ti != n_targets ; ++ti)
    for (size_t ioi = 0 ; ioi != interpolation_offsets[ti].size() ; ++ioi)
      if (green_offset_from_grid_offset[interpolation_offsets[ti][ioi]] == 0)
        green_offset_from_grid_offset[interpolation_offsets[ti][ioi]] =
          linear_index_from_stride_and_multi_index(
            green_stride, interpolation_grid_coordinates[ti][ioi]) ;

vector<int> target_green_offset ;
vector<size_t> target_grid_offset ;

// Subtract the nearby grid based interactions for each source.
for (size_t si = 0 ; si != n_sources ; ++si) {
  size_t const n_interactions = interaction_list[si].size() ;
  size_t const n_projection_points = projection_coefficients[si].size() ;

  // Map the projection stencil grid offsets to offsets on the mini grid
  // where the kernel values where cached.
  // psi -- projection stencil index
  projection_green_offsets.resize(n_projection_points) ;
  for (size_t psi = 0 ; psi != n_projection_points ; ++psi)
    projection_green_offsets[psi] =
      linear_index_from_stride_and_multi_index(
        green_stride,
          projection_grid_coordinates[si][psi]) - green_origin_offset ;

    // Determine the projected kernel values that need to be calculated.
    target_grid_offset.resize(0) ;
    target_grid_offset.reserve(
      n_interactions * ((n_interactions != 0) ?
        interpolation_offsets[interaction_list[si][0]].size() : 0)) ;

     // ii -- interaction index
     for (size_t ii = 0 ; ii != n_interactions ; ++ii) {
       size_t const ti = interaction_list[si][ii] ;
       copy(interpolation_offsets[ti].begin(), interpolation_offsets[ti].end(),
         back_inserter(target_grid_offset)) ;
     }

     sort(target_grid_offset.begin(), target_grid_offset.end()) ;
     vector<size_t>::iterator new_target_grid_offset_end = unique(
       target_grid_offset.begin(), target_grid_offset.end()) ;

     // Convert grid offsets to green offsets
     size_t num_unique_projection_points = new_target_grid_offset_end
       - target_grid_offset.begin() ;
     target_green_offset.resize(num_unique_projection_points) ;
     vector<size_t>::iterator grid_offset_it = target_grid_offset.begin() ;
     vector<int>::iterator green_offset_it = target_green_offset.begin() ;
     while (grid_offset_it != new_target_grid_offset_end)
       *green_offset_it++ = green_offset_from_grid_offset[*grid_offset_it++] ;

     // Scatter
     project_on_grid(
       target_grid_offset.begin(),
       new_target_grid_offset_end,
       target_green_offset.begin(),
       projection_green_offsets.begin(),
       projection_green_offsets.end(),
       projection_coefficients[si].begin(),
       kernel_values,
       projected_kernel_values) ;

     // Gather
     // For each interacting target interpolate the projected kernel values.
     // ii -- interaction index
     for (size_t ii = 0 ; ii != n_interactions ; ++ii) {
       // ti -- target index
       size_t const ti = interaction_list[si][ii] ;
       interaction_value_type & accumulator = interaction_values[si][ii] ;

       // isi -- interpolation stencil index
       size_t const n_interpolation_points = interpolation_offsets[ti].size() ;
       for (size_t isi = 0 ; isi != n_interpolation_points ; ++isi)
       multiply_subtract(
         accumulator,
         projected_kernel_values[interpolation_offsets[ti][isi]],
         interpolation_coefficients[ti][isi]) ;
     }
  }
}
```

It is possible that further performance improvements may be achieved by zeroing out parts of the Green's function that are likely to have to be precorrected later.

Precorrection with image sources

If translational invariance is used to cache the Green's function values, the precorrection for image panels and image Green's functions must be considered separately from precorrection for "direct" source kernels. If the precorrection for the image terms and the direct terms is performed separately, it might be worthwhile to setup a separate nearby interaction lists for the direct panels and for the image panels. Separating the nearby interaction lists would very significantly reduce the cost of precorrection for the image sources but it would require having two precorrection matrices or a method to combine the entries of the image precorrection matrix with the entries of the direct precorrection matrix. Note that, regardless of the position of the source panel and the test point (or test panel), the image panel is always further away from the test point (or test panel) than the original source. Therefore, the precorrection matrix for the image panels will always have a sparse structure that is a subset of the structure for the precorrection matrix for the direct panels. If the nonzero structure of the image precorrection matrix is a subset of the nonzero structure of direct precorrection matrix it is very likely that `subtract_grid_based_interactions` can be modified, or passed in a set of adequate iterators, such that combining the two matrices can be done efficiently and seamlessly. Developing this *split precorrection* approach is left as future work.

7.5 Calculating the image transform from a signal transform

When applying the precorrected FFT algorithm to problems involving *image sources*, such as the Stokes substrate Green's function or the Green's function for electrostatics in the presence of a ground plane, the projection of the source panels on to the FFT grid and the projection of the image panels on the FFT grid as well as their DFTs must be computed. In this section, a method for computing the image projection and its transform from their "direct" counterparts, without requiring an extra FFT, is presented.

Assuming that the plane of symmetry for defining image sources is the $z = 0$ plane and that the projection coefficients for the sources are $c_{m,n,p}$, the projection coefficients of the image sources are given by $d_{m,n,p} = c_{m,n,\mathrm{mod}(N_z-p,N_z)}$. Similarly, if the DFT of c is C then $D_{m,n,p} = C_{m,n,\mathrm{mod}(N_z-p,N_z)}$. However, if c is real its DFT C can be *compressed* using conjugate symmetry, i.e. it may be truncated such that it only has $\lfloor N_z/2 \rfloor + 1$ entries along the z direction. The Matlab code below illustrates how to produce a *compressed* D from a *compressed* C for 2D and for 3D.

Listing 7.8: *Compressed image transform from compressed signal transform in 2D.*

```
c = rand(M, N) ;
C = fftn(c) ;
C = C(:, 1:floor(N/2)+1) ;

D_from_C = C ;
D_from_C(2:end,2:end) = D_from_C(end:-1:2,2:end) ;
D_from_C(:, 2:end) = conj(D_from_C(:, 2:end)) ;

% D is the compressed image transform that we want, we can compare it to
$ D_from_C to validate the procedure.
d = c(:, [1 end:-1:2]) ;
D = fftn(d) ;
D  = D(:, 1:floor(N/2)+1) ;
```

Listing 7.9: *Compressed image transform from compressed signal transform in 3D.*

```
P = 6 ; M = 6 ; N = 6 ;
c = rand(P, M, N) ;
C = fftn(c) ;
C = C(:, :, 1:floor(N/2)+1) ;

D_from_C = C ;
D_from_C(2:end,:,2:end) = D_from_C(end:-1:2,:,2:end) ;
D_from_C(:,2:end,2:end) = D_from_C(:,end:-1:2,2:end) ;
% Note that the two steps above are not equivalent to
% D_from_C(2:end,2:end,2:end) = D_from_C(end:-1:2,end:-1:2,2:end)
D_from_C(:,:,2:end) = conj(D_from_C(:,:,2:end)) ;

% D is the compressed image transform that we want, we can compare it to
$ D_from_C to validate the procedure.
d = c(:, :, [1 end:-1:2]) ;
D = fftn(d) ;
D  = D(:, :, 1:floor(N/2)+1) ;
```

7.6 Preconditioning

To improve convergence of the iterative solver for (2.23) the block preconditioner from [55] was adapted to work with the Stokes flow Green's functions. The maximum block size is a user controllable parameter that can be used to trade setup time and memory for improved convergence.

Chapter 8

Conclusions and future work

In this chapter, the conclusions that were drawn in previous chapters are summarized and directions for future work are proposed.

Conclusions

A precorrected FFT accelerated algorithm for solving Stokes flow problems in the presence of a substrate was developed and demonstrated. Techniques to extend the applicability of the pFFT algorithm to certain types of non-translation invariant Green's functions were developed. The modified pFFT algorithm was validated against known theoretical, experimental and computational results and its performance was compared with previously published results.

Using the implicit substrate representation was shown to produce more accurate results with less memory and significantly less time than explicitly representing the substrate. Using the implicit substrate representation produces more accurate results because it accounts for the presence of the substrate exactly.

Surprisingly, a disappointing outcome of this study was that out-of-plane motion excites equation modes that reveal the need to refine the structure discretization as the distance to the substrate decreases. Simulation of out-of-plane motion also revealed that, when using an explicit substrate, the substrate discretization must be refined faster than than structure discretization for results to match the results obtained using implicit substrate discretization. So the implicit substrate representation has benefits but does not entirely decouple structure discretization from distance to the substrate.

An analytical panel integration algorithm for polynomial force distributions over odd powers of the distance between points on a flat panel and an evaluation point was developed extending previous results in the area.

Most of the blocks of the precorrected FFT algorithm where implemented using C++ template metaprogramming techniques that will facilitate the future development of accelerated boundary element solvers.

A stable velocity implicit time stepping scheme coupling the precorrected FFT solver with rigid body dynamics was introduced and demonstrated. The ODE library [27] was integrated with the solver to enable the simulation of systems with collisions, contacts and friction. Several techniques for speeding up the calculation of each time step were presented and tested. The time integration algorithm was found to produce reasonable results. However, it was found that some work still needs to be done to improve the robustness of the support for collisions and contacts of moving objects described by arbitrary meshes.

Directions for future work

In the future it would be interesting to couple membrane models such as those described in [47], [26] and [1] with the velocity-implicit integration scheme and the pFFT accelerated Stokes boundary element solver.

Since, for the microfluidic application examples, the panel distribution is not very homogeneous, coupling the velocity implicit stepping scheme with a fast solver that can better deal with non-homogeneous problems, such as the multipole method [33][20], could prove useful. Another alternative would be to develop a multi-resolution pFFT algorithm.

It would be very useful to integrate the solver with a scripting language such as Matlab, Python or Lua. Although this integration cannot be considered as research work it would greatly enhance the usability and the flexibility of the solver.

Another possibility for improvement would be supporting higher order panel force distributions to reduce the number of panels and improve convergence. Again this would not be considered to be very interesting research as the techniques for doing this are already developed. However, supporting higher order panels and force distributions would be useful for simulating smooth structures and would also facilitate the integration of the boundary element solver with a finite element solver for structural mechanics where high order shape and force distributions are commonly used.

Finally it would be both interesting and useful to further explore the techniques used for dealing with contacts, collisions and friction in order to provide better handling of complicated collision situations and to enable the time domain simulation using large time steps in the presence of multiple ongoing contacts. Still in this context, it would be worthwhile to try using multi-rate simulation techniques to speed up the simulation of systems where there are multiple moving objects, possibly undergoing collisions.

Bibliography

[1] C. Pozrikidis. *Boundary integral and singularity methods for linearized viscous flow*. Cambridge texts in applied mathematics. Cambridge University Press, 1992.

[2] William M. Deen. *Analysis of transport phenomena*. Topics in chemical engineering. Oxford University Press, 1998.

[3] Stephen D. Senturia. *Microsystem design*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[4] Y. H. Cho, A. P. Pisano, and R. T. Howe. Viscous damping model for laterally oscillating microstructures. *Journal of Microelectromechanical Systems*, 3:81–87, June 1994.

[5] Kwok P. Y., Weinberg M. S., and Breuer K. S. Fluid effects in vibrating micromachined structures. *Journal of Microelectromechanical Systems*, 14(4):770–781, August 2005.

[6] Ye Wenjing, Xin Wang, Werner Hemmert, Dennis Freeman, and Jacob White. Air damping in laterally oscillating microresonators: A numerical and experimental study. *Journal of Microelectromechanical Systems*, 12(5):557–566, October 2003.

[7] Lijie Li, G. Brown, and D. Uttamchandani. Air-damped microresonators with enhanced quality factor. *Journal of Microelectromechanical Systems*, 15:822 – 831, August 2006.

[8] Dale A. Anderson, John C. Tannehill, and Richard H. Pletcher. *Computational fluid mechanics and heat transfer*. Hemisphere Publishing Corporation, McGraw-Hill, 1984.

[9] Rajat Mittal and Gianluca Iccarino. Immersed Boundary Methods. *Annual Review of Fluid Mechanics*, 37:239–261, January 2005.

[10] O. C. Zienkiewicz, R. L. Taylor, and P. Nithiarasu. *The Finite Element Method for Fluid Dynamics*. Butterworth-Heinemann, 2005.

[11] Ladyzhenskaya O. A. *The mathematical theory of viscous incompressible flow*. Gordon & Breach, 1969.

[12] Happel J. and Brenner H. *Low Reynolds number hydrodynamics*. Mantinus Nijhoff, 1973.

[13] Greengard L. and Rokhlin V. A new version of the fast multipole method for the Laplace equation in three dimensions. 6:229–270, 1997.

[14] Hackbusch W. and Nowak Z.P. On the fast Matrix multiplication in the boundary element method by panel clustering. *Numer. Math.*, 54:463–491, 1989.

[15] J. Phillips and J. K. White. A Precorrected-FFT method for Electrostatic Analysis of Complicated 3-D Structures. *IEEE Trans. on Computer-Aided Design*, 16(10):1059–1072, October 1997.

[16] A. Frangi. A fast multipole implementation of the qualocation mixed-velocity-traction approach for exterior stokes flows. *Engineering Analysis with Boundary Elements*, 29:1039–1046, 2005.

[17] G. Biros, Ying L., and D. Zorin. A fast solver for the stokes equations with distributed forces in complex geometries. *J. Comput. Phys.*, 193(1):3170348, 2004.

[18] Xin Wang. *FastStokes: A Fast 3-D Fluid Simulation Program for Micro-Electro-Mechanical Systems*. PhD thesis, MIT, June 2002.

[19] J. Tausch. Sparse BEM for potential theory and Stokes flow using variable order wavelets. *Computational Mechanics*, 32(4-6):312–318, 2003.

[20] Ying Lexing. *An Efficient and High-Order Accurate Boundary Integral Solver for the Stokes Equations in Three Dimensional Complex Geometries*. PhD thesis, New York University, May 2004.

[21] Dino Di Carlo, Nima Aghdam, and Luke P. Lee. Single-cell enzyme concentrations, kinetics, and inhibition analysis using high density hydrodynamic cell isolation arrays. *Analytical Chemistry*, 78(14):4925–4930, 2006.

[22] Dino Di Carlo, Liz Y. Wu, and Luke P. Lee. Dynamic single cell culture array. *Lab Chip*, 6:1445–1449, 2006.

[23] Dino Di Carlo and Luke P. Lee. Dynamic single-cell analysis for quantitative biology. *Analytical Chemistry*, 78:7918–7925, 2006.

[24] J.R. Rettig and A. Folch. Large-scale single-cell trapping and imaging using microwell arrays. *Analytical Chemistry*, 77:5628–5634, 2005.

[25] Anil. K. Vuppu, Sanjoy K. Saha Antonio A. Garcia, Patrick E. Phelan, Mark A. Hayes, and Ronald Calhoun. Modeling microflow and stirring around a microrotor in creeping flow using a quasi-steady-state analysis. *Lab Chip*, 4:201–208, 2004.

[26] A. Beskok G. Karniadakis and N. Aluru. *Microflows and Nanoflows, Fundamentals and Simulation*, volume 29 of *Interdisciplinary Applied Mathematics*. Springer, 2005.

[27] Russell Smith. Open dynamics engine - user guide. www.ode.org, February 2006.

[28] David Abrahams and Aleksey Gurtovoy. *C++ Template Metaprograming - Concepts, tools and techniques from Boost and beyond*. Adison-Wesley, 2005.

[29] Andrei Alexandrescu. *Modern C++ Design*. Adison-Wesley, 2001.

[30] Frangi A. and Tausch J. A quallocation enhanced approach for stokes flow problems with rigid-body boundary conditions. *Engrg. Analysis Boundary Elements*, 29:886–893, 2005.

[31] J. N. Newman. Distribution of sources and normal dipoles over a quadrilateral panel. *Journal of Engineering Mathematics*, 20:113–126, 1986.

[32] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7:856–869, 1986.

[33] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[34] Stephen D. Senturia, Narayan Aluru, and Jacob White. Simulating the behavior of MEMS devices: Computational methods and needs. *IEEE Computational Science & Engineering*, 4(1):30–43, /1997.

[35] N.R. Aluru and J. White. A fast integral equation technique for analysis of microflow sensors based on drag force calculations. In *International Conference on Modeling and Simulation of Microsystems, Semiconductors, Sensors and Actuators*, pages 283–286, Santa Clara, April 1998.

[36] Ronald Cools. An encyclopedia of cubature formulas. *Journal of Complexity*, 19(3):445–453, June.

[37] J. L. Hess and A. M. O. Smith. Calculation of potential flow about arbitrary bodies. *Progress in Aeronautical Sciences*, 8:1–138, 1967.

[38] S. M. Rao, A. W. Glisson, D. R. Wilton, and B. S. Vidula. A simple numerical solution procedure for statics problems involving arbitrary-shaped surfaces. *IEEE Transactions on Antennas and Propagation*, 27:604–608, 1979.

[39] D. R. Wilton, S. M. Rao, A. W. Glisson, D. H. Schaubert, O. M. Al-Bundak, and C. M. Butler. Potential integrals for uniform and linear source distributions on polygonal and polyhedral domains. *IEEE Transactions on Antennas and Propagation*, 32:276–281, 1984.

[40] L. Knockaert. A general gauss theorem for evaluating singular integrals over polyhedral domains. *Electromagnetics*, 11:269–280, 1991.

[41] Charles F. Van Loan. *Computational Frameworks for the Fast Fourier Transform*. SIAM Publications, 1992.

[42] M. E. O'Neill. A slow motion of viscous liquid caused by a slowly moving solid sphere. *Mathematika*, 11:67–74, 1964.

[43] Howard Brenner. The slow motion of a sphere through a viscous fluid towards a plane surface. *Chem. Engrg. Sci.*, 16:242–251, 1961.

[44] Ye Wenjing, Joe Kanapka, and Jacob White. A fast 3d solver for unsteady stokes flow with applications to micro-electro-mechanical systems. In *Proceedings of the Second International Conference on Modeling and Simulation of Microsystems*, pages 518–521, San Juan, April 1999.

[45] Carlos Pinto Coelho, Luís Miguel Silveira, and Jacob K. White. A precorrected fft algorithm for stokes flow in the presence of a substrate. *(submitted to) Journal of Microelectromechanical Systems*, 2007.

[46] Hairer E., Lubich C., and Wanner W. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Number 31 in Springer series in computational mathematics. Springer, 2004.

[47] C. Pozrikidis, editor. *Modeling and simulation of capsules and biological cells*. CRC Mathematical Biology and Medicine Series. Chapman & Hall, 2003.

[48] David Baraff. An introduction to physically based modeling: Rigid body simulation I - unconstrained rigid body dynamics. Online Siggraph '97 Course notes, Carnegie Mellon University - Robotics Institute.

[49] S. A. Sheynin and A. V. Tuzikov. Explicit formulae for polyhedra moments. *Patter Recognition Letters*, 22:1103–1109, 2001.

[50] R. Barzel and A. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22(4), August 1988.

[51] David Baraff. An introduction to physically based modeling: Rigid body simulation II - nonpenetration. Online Siggraph ’97 Course notes, Carnegie Mellon University - Robotics Institute.

[52] Andrew Witkin. An introduction to physically based modeling: Constrained dynamics. Online Siggraph ’97 Course notes, Carnegie Mellon University - Robotics Institute.

[53] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[54] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. special issue on ”Program Generation, Optimization, and Platform Adaptation”.

[55] J. Tausch and J. White. Preconditioning first and second kind integral formulations of the capacitance problem. In *Proceedings of the 1996 Copper Mountain Conference on Iterative Methods*, April 1996.