Differentiable Simulation Methods for Robotic Agent Design

by

Tao Du

B.Eng., Tsinghua University (2013)
M.S., Stanford University (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Signature of Author . . . .
Department of Electrical Engineering and Computer Science
August 27, 2021

Certified by. . . . .
Wojciech Matusik
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Differentiable Simulation Methods for Robotic Agent Design

by

Tao Du

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Designing robots with extreme performance in a given task has long been an exciting research problem drawing attention from researchers in robotics, graphics, and artificial intelligence. As a robot is a combination of its hardware and software, an optimal robot requires both an excellent implementation of its hardware (e.g., morphological, topological, and geometrical designs) and an outstanding design of its software (e.g., perception, planning, and control algorithms). While we have seen promising breakthroughs for automating a robot's software design with the surge of deep learning in the past decade, exploration of optimal hardware design is much less automated and is still mainly driven by human experts, a process that is both labor-intensive and error-prone. Furthermore, experts typically optimize a robot's hardware and software separately, which may miss optimal designs that can only be revealed by optimizing its hardware and software simultaneously.

This thesis argues that it is time to rethink robot design as a holistic process where a robot's body and brain should be co-optimized jointly and automatically. In this thesis, we present a computational robot design pipeline with differentiable simulation as a key player. We first introduce the concept of computational robot design on a real-world copter whose geometry and controller are co-optimized with a differentiable simulator, resulting in a custom copter that outperforms designs suggested by human experts by a substantial margin. Next, we push the boundary of differentiable simulation by developing advanced differentiable simulators for soft-body and fluid dynamics. Contrary to traditional belief, we show that deriving gradients for such intricate, high-dimensional physics systems can be both science and art. Finally, we discuss challenges in transferring computational designs discovered in simulation to real-world hardware platforms. We present a solution to this simulation-to-reality transfer problem using our differentiable simulator on an example of modeling and controlling a real-world soft underwater robot. We conclude this thesis by discussing open research directions in differentiable simulation and envisioning a fully automated computational design pipeline for real-world robots in the future.

Thesis Supervisor: Wojciech Matusik
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

When I came to MIT six years ago with a Master's degree, I thought I already knew how to survive and thrive in a graduate school. I thought I was ready for a smooth journey towards a Ph.D. degree. However, it quickly turned out that my graduate school experience was much wilder and more unpredictable than I had ever expected. It has been a journey full of surprises, excitement, challenges, and difficult times. It is a journey with both Eureka moments and sleepless nights from time to time. Nevertheless, it is an undoubtedly rewarding journey that, after six years, leads to this thesis work, which would not be possible without the help I have received from countless people.

First and foremost, I would like to thank my Ph.D. advisor, Prof. Wojciech Matusik, for teaching me everything about being a good researcher and mentor. Wojciech always encourages me to think big and aim high. He urges me to solve ambitious problems and reach high research standards but never leaves me helpless. I thank Wojciech for spending many hours enlightening me with new research ideas, being tolerant of our occasional debates on research directions, and inspiring me to reach my potential whenever I feel aimless or hopeless in a project.

I would also like to thank Prof. Daniela Rus, Prof. Armando Solar-Lezama, and Prof. Justin Solomon for serving on my thesis committee. I thank Daniela for sharing her broad knowledge and deep insights in soft robotics and always encouraging me to "keep the gradients!". I thank Armando for introducing the power and elegance of program synthesis to me, which I am still benefitting from in my research these days. Finally, although Justin was not directly involved in my thesis work, I thank him for teaching me all the numerical methods in this thesis and mentoring me with patience and support back in the old days when I was a Master's student.

I am also thankful to other professors who have offered their generous help during my Ph.D. time. Prof. Bernd Bickel mentored me in my very first SIGGRAPH project with constant guidance and constructive feedback. Prof. Eftychios Sifakis taught me the fundamentals of finite element methods. I thank Eftychios for giving me informative and enjoyable sessions about numerical analysis, partial derivative equations, and fluid simulation, which I will always cherish. I am also thankful to my academic advisor, Prof. Polina Golland, for offering her emotional support and career advice whenever I need them.

I was extremely fortunate to have three outstanding labmates who mentored me and shaped how I conducted research when I was a junior Ph.D. student: Bo Zhu, Adriana Schulz, and Desai Chen. As a postdoctoral researcher in our lab, Bo taught me how to develop good research taste and seek fundamental problems. Adriana, my academic sister, is my role model of a good teacher, mentor, and collaborator. Finally, my academic brother Desai helped me develop critical thinking and rigorous mathematical and numerical skills needed in graphics research.

I also want to thank the rest of the current and former members of our lab: Vahid Babaei, Nick Bandiera, Timothy Erps, Michael Foshey, Alexandre Kaspar, Petr Kellnhofer, Changil Kim, David Kim, Mina Konaković Luković, David Levin, Beichen Li, Yifei Li, Yiyue Luo, Pingchuan Ma, Liane Makatura, James Minor, Tae-

Hyun Oh, Javier Ramos, Liang Shi, Wan Shou, Mélina Skouras, Andrew Spielberg, Subra Sundaram, Yunsheng Tian, Sebastien Wah, Harrison Wang, Kui Wu, Jie Xu, Allan Zhao, and Lara Zlokapa. It has been a privilege to work with many of you during sleepless nights before SIGGRAPH deadlines. My special thanks go to Michael Foshey for inviting Buttercup to join the lab as a mascot, Andrew Spielberg for being a long-time collaborator, a personal friend, and a good StarCraft player, and Kui Wu for always urging me to polish up every paper we have co-authored. Additionally, I thank Britton Bradley and Roger White for their incredible administrative work supporting every aspect of our research.

My Ph.D. would be incomplete without support from my collaborators outside our lab: Hang Chu, Chuang Gan, Yuanming Hu, Zhiao Huang, Josie Hughes, Jeevana Priya Inala, Robert Katzschmann, Joseph Lambourne, Jieliang Luo, Yewen Pu, Hao Su, Joshua Tenenbaum, Karl Willis, John Zhang, and Siyuan Zhou. I thank them for all the good and bad moments we experienced together in every paper we co-authored.

I would also like to thank sincerely Prof. Vladlen Koltun and Dr. Qianyi Zhou for mentoring me when I was a Master's student at Stanford University. Both of them taught me so much about the fundamentals of doing high-quality research with their first-hand experience, which still helps me today whenever I face new research problems. I am also grateful to Prof. Bin Wang and Prof. Jun-Hai Yong for introducing to me the world of computer graphics when I was an undergraduate student.

Last but not least, I am incredibly grateful to my friends and my family for their company and support. I thank Ruoyu Wang for being a long-time friend since my undergraduate years and for the joyful moment when we together beat two extreme AIs in Age of Empires 2 for the first time. I thank Caiyue and Jiusi for introducing their furry and fluffy friends Jiuge and Xiangxiang to my wife and me. I am also grateful to my soccer teammates for all our games on MIT Roberts field on midsummer nights. Furthermore, I owe many thanks to my parents for their unconditional love, support, and understanding whenever I face life-changing decisions. Finally, I would like to thank my wife and lifelong best friend, Yingzhe, for always believing almost blindly in my potential whenever I struggle with research, for kindly allowing me to join meetings during our vacation in the England countryside, and for unconditionally offering her love and company whenever I need them. This thesis is dedicated to her and our parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Designing a robot with optimal performance for a given task has long been an exciting research problem in robotics, graphics, and embodied intelligence. Finding an optimal robot requires a deep understanding of its body and brain, i.e., a good robot design must consist of both good hardware and good software. Many traditional computational methods focus on exploring the software space only, e.g., via mathematical tools and numerical algorithms developed for perception, sensing, planning, and control. However, these methods typically overlook the need to design the geometry, morphology, or topology of robotic agents computationally: Today, it is still common to ask an experienced mechanical engineer instead of a computational design algorithm to drive a robot's hardware design manually, which is often labor-intensive and error-prone.

In this thesis, we argue that this human-guided design process suffers from two significant drawbacks: First, the robot's hardware and software are optimized separately, which may miss optimal designs that can only be discovered by joint optimization of both. Second, and more importantly, as engineers drive the whole design process, its quality relies heavily on one's domain-specific knowledge or prior design experience. *Can computational methods outperform human experts in designing a robot?* This is the core question we ask and attempt to answer in this thesis.

1.1 Traditional Robot Design

We first review the basic idea behind a traditional robot design process before discussing the core techniques we developed in this thesis. Consider designing an aerial robot that aims to carry as much payload as possible (Fig. 1-1). We start by parametrizing the design of a robot using discrete and continuous parameters. For example, discrete parameters may include the number of actuators, the topological structure, and the control hierarchy, and continuous parameters can include the geometrical size of a component, the material properties, and coefficients in a controller. Traditionally, an engineer first proposes an educated guess of the continuous and discrete parameters that determine an initial hardware and software prototype. She may then evaluate the proposed design in simulation, analyze its results, and make adjustments to both discrete and continuous design parameters. For example, with the goal of maximizing the payload in mind, she may propose using lighter and wider foam wings based on her domain-specific knowledge or past design experience. Once she is content with the simulation results, she builds a hardware prototype, collects data from real-world experiments, and makes further adjustments to the design parameters as well as the simulation model based on feedback from both simulation and experiments. This process typically iterates multiple times before converging to an optimal design. As can be seen from the description above, the whole procedure is mainly driven by the engineer's experience and expertise.

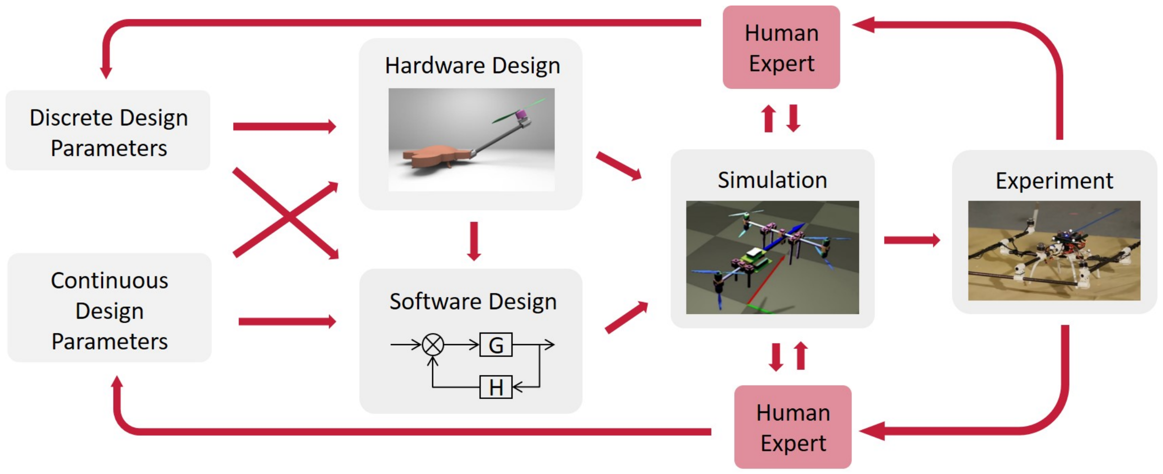


Figure 1-1: An example of designing a real-world multicopter. Human experts (red blocks) play a central role in iterating the design.

1.2 Thesis Overview

This thesis argues that it is now the right time to revolutionize this human-in-the-loop design process by making computational methods a first-class citizen. With the recent development of more powerful computing platforms and more advanced simulation, optimization, and learning algorithms, this thesis focuses on automating a subset of the design process in Fig. 1-1 with the following simplification: First, we skip the exploration of discrete parameters and focus on optimizing continuous design parameters. Throughout the thesis, we will assume all discrete parameters are given and fixed by a user. Second, we limit the scope of hardware and software optimization to continuous shape and control parameters only. Other hardware or software components like sensors, actuators, or perception algorithms are beyond the scope of this thesis. We provide an overview of the computational robot design pipeline with the simplifications described above in Fig. 1-2, and we will briefly discuss our other research work that can help to remove these simplifications at the end of this thesis.

The core technique in this thesis is the development and usage of differentiable simulation, a recent simulation methodology that we believe can be a crucial player in

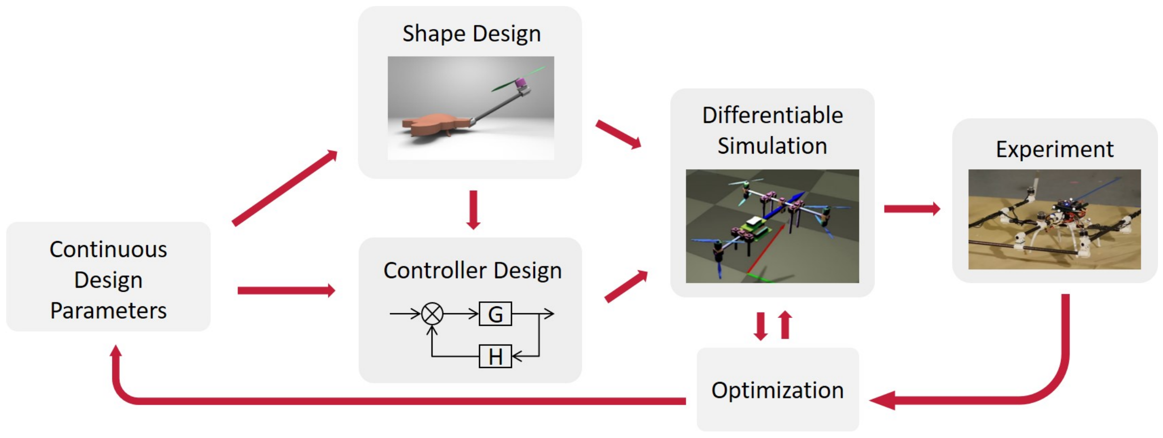


Figure 1-2: A computational robot design pipeline covered in this thesis. Compared with Fig. 1-1, we skip the discrete design parameters and focus on optimizing shape and control designs. Also, note the central role of differentiable simulation in the pipeline.

realizing a computational robot design pipeline. Unlike traditional simulation tools that only evaluate a robot's performance passively, a differentiable simulator connects a robot to its performance by backpropagating the gradient of a performance metric to the continuous shape and control parameters. Such gradient information unlocks continuous optimization algorithms for the continuous shape and control design space. In the first part of this thesis (Chpt. 3), we will demonstrate this computational design pipeline and motivate the usage of a differentiable simulator using a case study on real-world multicopters [43]: In this work, we build a differentiable rigid-body simulator and leverage its shape and control gradients to explore various geometric designs of functional copters effectively. The new copter design discovered by our computational pipeline is then evaluated in real-world flight tests, in which we observe substantial improvement (e.g., 30% increase in its maximum payload) in its performance compared with a manual design created by human engineers.

The success in multicopter design shows the power and potential of a computational pipeline equipped with a differentiable simulator in real-world robot agent design problems. To extend this pipeline to supporting more complex robots, we introduce two advanced differentiable simulators for more intricate and higher-dimensional physics systems. In the second part of this thesis (Chpt. 4), we present a differentiable soft-body simulator [44] and a differentiable fluid simulator [45] with tens of thousands of degrees of freedom. Contrary to the common belief that adding gradients to a physics simulator requires merely a procedural application of existing mathematical tools (e.g., the chain rule, the adjoint method, and sensitivity analysis), we use these two advanced differentiable simulators to show that deriving gradients can be both a science and an art. Specifically, by identifying the numerical pattern in the governing equations of these systems, we show that we can obtain substantial speedup and simplicity from carefully crafted numerical solvers. Furthermore, the speedup paves the way for us to extend our pipeline [43] to soft and underwater robots because simulation and gradient computation in these robots are the typical computational bottleneck.

The more powerful soft-body and fluid differentiable simulators we have developed unlock exciting soft and underwater robotics applications. Moreover, these physics-based simulation tools pave the way for transferring design results in simulation to reality, known as the simulation-to-reality (sim-to-real) task. In the last part of this thesis (Chpt. 5), we will present a solution to the sim-to-real task and demonstrate it on a real-world soft underwater robot modeling and control problem [41] that we manage to solve with the help of our advanced differentiable simulator [44]. In particular, we present a Starfish robot and show that we can use our differentiable simulator to jointly identify its system parameters and propose effective open-loop control signals.

1.3 Contributions

This thesis contributes the following:

- A computational co-design and co-optimization pipeline that jointly optimizes a rigid robot's shape and control with a demonstration on real-world multicopter design problems [43];
- A physics-based, efficient, robust soft-body differentiable simulator [44] and its applications in system identification, trajectory optimization, and closed-loop control;
- A physics-based, differentiable Stokes-flow fluid simulator [45] and its application in designing functional fluidic devices;
- A computational, differentiable-simulation-based method for narrowing sim-to-real gaps and its application in modeling and controlling a real-world soft underwater robot [41].

1.4 Thesis Impact

We believe that the computational methods developed in this thesis and their future directions can fundamentally change the way engineers build robots in the future. The traditional robot design, optimization, and fabrication processes are mainly manual, with computational methods playing a secondary role. The computational robot design pipeline presented in this thesis provides a digital solution that is faster and less error-prone than the traditional, human-guided design process. We demonstrate in this thesis the efficacy of this computational design pipeline in co-designing and co-optimizing multicopters, and we believe that designing other types of rigid robots like quadrupeds or manipulators can also benefit from our pipeline.

A second positive impact that our thesis brings is the development of advanced differentiable simulation methodology. Without a fast and robust differentiable simulator, it would not be possible to apply the computational robot design pipeline to more sophisticated robotic systems with high DoFs, e.g., soft manipulators or soft underwater robots. The efficient differentiable simulators developed in this thesis tackle this exact problem, unlocking the full power of computational methods in downstream applications relevant to designing soft robots, e.g., system identification, motion planning, and controller design.

Lastly, this thesis provides a new, differentiable-simulation-based method for narrowing the sim-to-real gap when transferring computationally designed robots in simulation to their hardware platforms. In particular, the efficient differentiable simulator developed in this thesis makes it tractable to identify the system parameters and narrow the reality gap of a soft underwater robot, bringing new opportunities in soft robot design, control, optimization, and fabrication.

We hope that the differentiable simulators developed in this thesis [43, 44, 45], along with the proposed computational design pipeline [43] and the discussion on sim-to-real transfer [41], will push the frontier of robot design in multi-physics environments and unlock new opportunities in upstream and downstream applications, including shape representation and analysis in computer-aided design (CAD) [42], morphological design of soft robots [99], robotic planning and control [44], and multi-objective design and optimization [98].

Chapter 2

Related Work

This chapter reviews previous papers from the three most relevant topics to our thesis: computational robot design (Chpt. 3), physics simulation (Chpt. 4), and sim-to-real transfer (Chpt. 5).

2.1 Computational Robot Design

As we discussed in the previous chapter, our computational robot design pipeline focuses on jointly optimizing a robot's shape and controller design. Here, we review previous papers on computational robot design and classify them into three categories: shape design, controller design, and co-design of both shape and control.

2.1.1 Shape Design

Related work on the computational design of a robot's shape can be traced back to early papers about the computational fabrication of static and dynamic objects without controllers, e.g., designing plush toys [109], furniture [122, 146, 83, 127], clothes [147], inflatable structures [133], wire meshes [54], mechanical objects [79, 166, 31], and masonry structures [158, 151]. Previous papers have also discussed the computational design of objects with more dynamic motions [8].

Designing a robot's shape is typically much more complex than the papers above because of the existence of control signals over a long time horizon. Therefore, previous papers about shape optimization typically apply their methods to passive dynamical systems. For example, Umetani et al. [148] present a system that allows the user to design the shape of free-form hand-ranching gliders. Martin et al. [102] present a data-driven approach to capture parameters of an omnidirectional aerodynamics model, which is then used to design three-dimensional kites. Bharaj et al. [17] present a method that leverages physical simulation and evolutionary optimization to refine the mechanical designs of automata such that the resulting toys can walk.

Compared with designing a rigid robot's shape, designing the shape of a soft robot is even more challenging because of its large degrees of freedom (DoFs), and related work is much sparser. One common strategy is to run genetic algorithms to evolve a voxel-based soft robot design in simulation [66], leading to computer-designed soft robots that can walk, swim, and grow [29, 32, 20]. The strategy in this thesis is quite different from these works in that we rely on differentiable simulation and gradient-based optimization methods to design a robot, which is typically much more sampling efficient.

2.1.2 Controller Design

Compared with shape design, designing a robot's controller is more automatic and can enjoy many well-developed tools and techniques. Previous papers have proposed many computational methods for designing, editing, and optimizing control signals in systems like walking robots [152, 30], mechanical characters [31], swimming characters [88, 141], birds [160, 76], and gliders [148]. Particularly, for the sake of efficiently generating artist-desired animations, a large variety of optimization algorithms have been developed in the effort to edit and control the dynamics of rigid bodies [118, 145, 74, 140, 88]. Apart from optimization-based methods, it is also worth mentioning the success of (reinforcement-)learning-based controller design in many real-world robots [162, 163, 72]. Optimizing a soft robot's controller is also covered in previous papers. e.g., controlling soft swimmers [107] or soft quadrupeds [15]. However, it is still an under-explored area because of the computational bottleneck from simulating and optimizing many DoFs in a soft robot.

2.1.3 Co-Design and Co-Optimization

Jointly optimizing a robot's shape and control is a challenging problem due to their intricate coupling. Therefore, perhaps not surprisingly, research on co-design and co-optimization of a robot is sparse and starts to draw people's attention only in recent years. Besides our work in Chpt. 3 and its related work, there are also a few other co-design and co-optimization strategies from the literature: Deimel et al. [37] present a method for co-designing a soft hand's morphology and control signals based on fast forward simulation. In addition, Wang et al. [156] represent robot designs as graph networks and co-optimizes a robot's topology and controller. Compared with their methods, this thesis takes a fundamentally different approach based on gradient-based optimization in a continuous domain of design parameters.

2.2 Physics Simulation

Simulation is a critical ingredient in our computational robot design pipeline. In Chpt. 4, we introduce differentiable soft-body and fluid simulators, which we build upon previous work on soft-body simulation, fluid simulation, and more recent development of differentiable physics simulation.

2.2.1 Soft-Body Simulation

Soft-body simulation is an extensively studied topic in computer graphics and mechanical engineering. The differentiable soft-body simulator we develop in Chpt. 4 is most relevant to Projective Dynamics (PD), a fast and implicit soft-body simulation methodology with specific assumptions on material models [22]. Existing works have extended standard PD algorithm to support rigid bodies [89], conserve kinetic energy [39, 40], and a wide range of hyperelastic materials [96]. Furthermore, previous papers have also proposed more advanced acceleration strategies, including semi-iterative Chebyshev solvers [153, 154], parallel Gauss-Seidel methods with randomized graph coloring [53], precomputed reduced subspace methods for the required constraint projections [24], and multigrid solvers [161]. Furthermore, Macklin et al. [100] introduce a preconditioned descent-based method of Projective Dynamics on GPU with a penalty-based contact model. Our differentiable simulation in Chpt. 4 inherits the standard PD setup. However, it is quite different from these works above in that we study how to leverage PD simulation techniques to speed up gradient computation, which is one of the unique contributions in this thesis.

Another central topic in soft-body simulation is collision handling. There exists a diverse set of collision handling algorithms with different design considerations: physical plausibility, time cost, and implementation complexity. Only a few of them [90, 28, 69, 100] take differentiability of a contact model into account. The most widespread strategy in PD-based simulation is to treat contact as soft constraints by either directly projecting colliding vertices onto collision surfaces at the end of each simulation step [153, 154, 40, 39] or imposing a fictitious collision energy [96, 89, 22]. Such methods introduce an artificial stiffness coefficient, which is task-dependent and requires careful tuning. Alternatively, Ly et al. [97] propose to combine the more physically plausible Signorini-Coulomb contact model with PD in forward simulation. However, using such a model creates an additional challenge during backpropagation as solving the contact forces requires nontrivial iterative optimizers, which either fail to maintain differentiability or no longer enjoy the speedup from PD. Our differentiable simulation in Chpt. 4 considers both penalty-based and complementarity-based contact models by drawing inspirations from sparse Cholesky updates [65] and low-rank matrix updates [75].

2.2.2 Fluid Simulation

Simulation of fluid flows has been a staple of physics-based animation, relying predominantly on the Navier-Stokes equations to capture the dynamics of motion in media such as smoke [137, 49] and water [48]. Several such methods are based on finite-difference discretizations on regular Cartesian grids, often with a staggered placement of state variables. Level-set methods [112] have been used widely to solve interface problems on a Cartesian grid, in conjunction with the numerical schemes to treat the boundary such as the Ghost Fluid Method [50] and variational interpolation [10]. Explicit boundary discretizations, such as embedded surface meshes, show their unique merits in modeling the sub-cell geometry and enforcing precise boundary conditions [125, 7]. These embedded discretizations of the variational type are focused on handling Dirichlet boundaries [168, 64], which inspired our discretization scheme in the differentiable fluid simulator presented in Chpt. 4. These discretizations can conveniently accommodate adaptive resolution [4] and flows in containers with deforming geometry [51]. Accommodation of changing geometry of the fluid container is also addressed in grid-based techniques that draw inspiration from Arbitrary Lagrangian-Eulerian (ALE) techniques [73].

Notably, most such works targeting evolving fluid domains focus on dynamic simulation rather than stationary flows. Steady-state flows, and especially Stokes fluids, have received occasional attention within the graphics literature, in applications related to fluid control [18], simulation of highly viscous media such as paint [11], or as a complement to an unsteady-flow solver for viscous liquids [82]. Our differentiable fluid simulator presented in Chpt. 4 falls in the category of steady-state Stokes flow.

Besides prior efforts on Stokes flow, our work in Chpt. 4 is also related to methods on simulating nonlinear compressible flows [81] and viscoplastic materials [138] but is different from them: Instead of solving the Navier-Stokes equations with explicit pressure terms, we exploit the analogy between Stokes flows and linear elasticity to simulate differentiable, quasi-incompressible Stokes flow without the need for solving the pressure term explicitly. The idea of drawing the analogy between fluids and elastic solids for fluid simulation can also be found in Ferst et al. [52], which simulates fluids on an adaptive octree grid using a hexahedral finite element discretization. Although both Ferst et al. [52] and our method leverages finite element discretization and elastic solvers for fluid simulation, their approach focuses on forward simulation only. Meanwhile, we develop differentiable flow simulation with comprehensive discussions on gradient derivation.

2.2.3 Differentiable Simulation

Many recent advances in differentiable physics facilitate applying gradient-based methods in robotics learning, control, and design tasks. Several differentiable simulators have been developed for rigid-body dynamics [117, 35, 143, 36, 100], soft-body dynamics [69, 68, 62, 56], cloth [93, 119], and fluid dynamics [144, 105, 159, 123, 67]. This thesis is most relevant to differentiable soft-body simulation methods (Chpt. 4), which we briefly discuss below.

ChainQueen [69] and its follow-up work DiffTaichi [68] introduce differentiable physics-based soft-body simulators using the material point method (MPM), which uses particles to keep track of the entire states of the dynamical system and solves the momentum equations as well as collisions on a background Eulerian grid. However, the explicit time integration in their methods requires small time steps to preserve numerical stability, leading to significant memory consumption during backpropagation. To resolve this issue, ChainQueen proposes to cache checkpoint steps in memory and reconstruct the states by rerunning forward simulation during backpropagation, which increases implementation complexity and introduces extra time cost. Further, solving collisions on an Eulerian grid may introduce artifacts depending on the resolution of the grid [63]. Another particle-based strategy using explicit time integration is to approximate soft-body dynamics with graph neural networks [92, 121], which is naturally equipped with differentiability but may result in physically implausible behaviors.

Unlike those above differentiable soft-body simulators using explicit time integration, the differentiable simulation introduced in Chpt. 4 employs fully implicit time integration. Therefore, our differentiable simulator in Chpt. 4 is more similar to Hahn et al. [62] and Geilinger et al. [56]. Compared with their work and other soft-body differentiable simulators [93, 119] that systematically apply a combination of the adjoint method and sensitivity analysis to derive gradients, we put a special focus on a more strategic backpropagation scheme that leverages the unique numerical properties in a physics system's governing equation, leading to empirically faster gradient computation.

2.3 Sim-to-Real Transfer

The sim-to-real transfer is crucial for any computational methods developed in simulation, and our computational robot design pipeline is not an exception. The solution presented in Chpt. 5 draws inspiration from previous papers on narrowing sim-to-real gaps [27, 142, 116, 162, 38]. Additionally, Chpt. 5 shares similarities with model-based deep reinforcement learning methods in [38] or [27], but we leverage full gradient information from an analytic dynamic model in simulation. Another commonly used strategy for closing the sim-to-real gap is domain randomization, which trains the controller with randomized models in simulation [142, 116]. Essentially, domain randomization attempts to absorb modeling errors by training a robust but conservative controller. Our method is different from this family of methods in that we attempt to reduce modeling errors by improving the model parameter estimation directly.

Chapter 3

Computational Robot Design

In this chapter, we introduce the basic idea behind a computational robot design pipeline [43]. We present the pipeline using a case study of designing a multicopter, a rigid robot with many industrial applications. The technical method discussed in this chapter covers a subset of the complete computational design pipeline in Fig. 1-2, which we highlight in Fig. 3-1 below. Specifically, this chapter focuses on computational methods that automatically optimize a multicopter's shape and controller designs in simulation. We then directly transfer the resultant designs to hardware platforms without further modification or discussions on the potential performance discrepancy due to sim-to-real gaps, which we will elaborate on in detail in Chpt. 5.

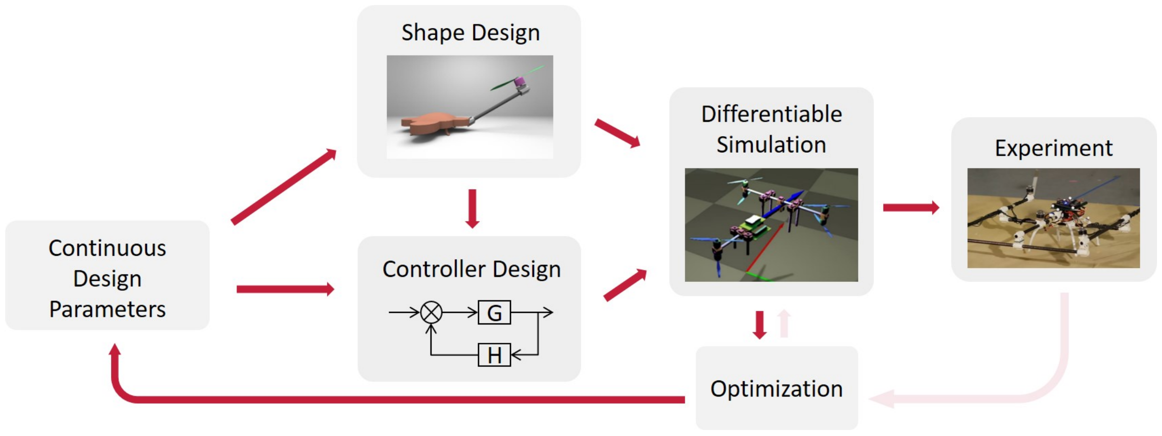


Figure 3-1: The computational multicopter design method discussed in this chapter. Note that methods for narrowing the sim-to-real gap are not covered here and will be discussed in later chapters.

3.1 Motivation

Multicopters are aerial vehicles that are becoming more and more popular. They are mechanically simple and can be controlled manually or automatically to have a stable and accurate motion. For this reason, these vehicles are increasingly being used in many settings, including photography, disaster response, search and rescue operations, hazard mitigation, and geographical 3D mappings. Most current multicopter designs are fairly standard (e.g., symmetric quadcopters or hexacopters). Designing a non-standard multicopter that is optimized for a specific application is challenging since it requires expert knowledge in aerodynamics and control theory.

In this chapter, we propose a design process that allows non-expert users to build custom multicopters that are optimized for specific design goals. Our system allows users to concentrate on high-level design while computation handles all the necessary elements to ensure the correct function of the resulting physical models. Our intuitive composition tool enables users to express their creativity and to explore trade-offs between different objectives in order to develop machines well suited for specific applications.

Our system also allows us to expand the design space of multicopters. Typical space encompasses a small set of standard designs with a symmetric distribution of rotors and all propellers oriented upright. In this design space, forces and torques induced by propellers are easily balanced and controllers can be easily adjusted. However, this does not allow the design of nonsymmetric multicopters that are more optimal for some specific tasks. For example, the field view of a camera in a hexacopter could be obstructed by the motors, so one might prefer removing a motor in front of the camera. Another example could be a multicopter carrying an irregularly shaped object like a wide-band antenna. In this case, an asymmetric design gives more freedom to specify mass distribution for better flying stability. Finally, a nonstandard design with extra motors can increase the payload and improve fault tolerance, which are key factors for product delivery. Our computational design employs parametric representations that capture variably in the general shape, rotor positions and orientations, and performs optimizations based on user-specified metrics. This enables users to explore the shape space and discover functional vehicles that significantly differ and outperform the standard models.

An immediate challenge after we expand the design space is to find a good controller for non-standard multicopter designs. Due to its uneven distribution of mass and inertia tensor as well as its random rotor position and orientation, applying a traditional controller from classic multicopters directly requires nontrivial and tedious parameter adjustments. Moreover, during flight, the performance of a multicopter is jointly influenced by its dynamics and control signals and therefore optimization has to include both shape variables and controller parameters. Finally, there is a tradeoff between allowing users to freely express metrics and still keeping the optimization problem tractable. In our system, we use Linear-Quadratic-Regulator (LQR), an optimal control method for nonstandard multicopter designs, and the controller is automatically determined to avoid tedious parameter adjustments. We formulate an optimization problem which includes both shape and control variables and propose an algorithm to effectively find the optimal shape as well as the control parameters for a given design. We specify the user metrics as a bi-convex function, which is expressive enough to represent many useful metrics.

To summarize, our main contributions in this chapter include:

- Providing a complete pipeline that allows users to design, optimize, and fabricate multicopters.
- Formulating an optimization problem that can jointly optimize the shape and control parameters according to different design metrics.
- Providing an efficient numerical scheme to solve the multicopter optimization problem and show its efficacy by optimizing various types of non-standard multicopter designs.

3.2 Geometry Design

Our interactive design tool allows users to compose new models from parts in a database, which contains standard parts like propellers, motors, carbon fiber rods, and a variety of free-form body frames. Though the database is relatively small, users can create a widely diverse set of designs by manipulating and composing them.

3.2.1 Part Database

Following the ideas in Schulz et al. [127], both functional parts and free-form body frames in the database are *parametric*, represented by a feasible parameter set and a linear mapping function that returns different geometries for different parameter configurations. The use of parametric shapes allows geometric variations while guaranteeing that all shape manipulations preserve structure and manufacturability. In our model, all parameter configurations in the feasible set are guaranteed to be manufacturable geometries. Linear maps were chosen because they speed up computation without compromising too much on expressiveness. In this linear representation, geometries are represented as meshes where each vertex is a linear function of the parameters.

Each part in the database is annotated with connecting patches that indicate regions where parts can be attached to each other. Since every part is parametric, the position of each patch is also a function of the parameters. We define different patch types for different composition methods. For example, carbon tubes are annotated with circular patches while the bottom of a rotor has a flat patch (see Fig. 3-2). Each patch type includes a parametric representation of its center and additional information for alignment (for example, circular patches include the radius and main axis and flat patches include the normal). Our collection was designed and parametrized by mechanical engineers.

3.2.2 Composition

After the user drags in a new part and calls the *connecting* operation, the system aligns the parts with respect to the working model and adds the appropriate connecting components. The two closest patches are used to create a connection between the components.

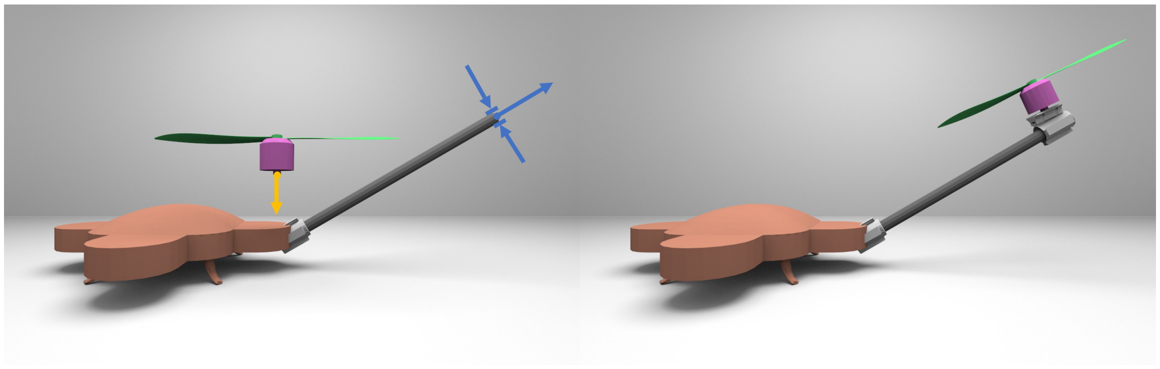


Figure 3-2: Example of composition. Left: parts with highlighted patches (circular patch with an annotated main axis and a diameter in blue and flat patch with an annotated normal in orange). Right: composed design.

We define a list of rules that designate appropriate connecting parts for each type of patch pair. Connecting parts are selected from a small list of standard components, e.g., patches of carbon fiber rods need circular adapters and plates need mounts with screws.

Next, we position parts and appropriate connectors relative to each other. Our goal is to preserve the parametric representation in the composed designs so that users can continue to manipulate the geometry at every step in a structure-preserving fashion. In the composed model the parameter set is the union of the parameters of the containing parts and the feasible set is the intersection of the feasible sets. Therefore, the alignment step must define constraints on part positions so that parts are correctly placed relative to each other in all feasible configurations. This is done in two steps: First, the patch information is used to appropriately rotate the models relative to each other (in the example in Fig. 3-2, the normal of the flat patch is made perpendicular to the main axis of the circular patch). Rotations are dealt with first because these are not linear operations and therefore cannot be represented by our linear parameter mapping. Since the position of each patch is represented as function of the parameters, alignment translations involve constraining these two functions to be equal. Therefore parts are aligned by adding constraints to the feasible set of the composed parametric design and solving for the closest feasible solution.

The parameters of the composed design are not only useful for allowing users to better explore the design space, but they also describe the possible variations for automatic design optimization. Since both parts and composition schemes are defined using linear models, linearity is preserved in the resulting composed designs, which include a set of equality and inequality constraints. Equality constraints can be removed by replacing the original shape parameters with free variables in the affine space defined by the linear equations. As a result, we will use $\mathbf{s}$ to denote the new shape parameters and $\mathbf{A}^{\text{ineq}}\mathbf{s} \leq \mathbf{b}^{\text{ineq}}$ to represent these constraints in later sections. The linear parameter variations allow parts of the copters to scale and move relative to each other but do not define local rotations. We therefore augment the feasible set by defining additional variables, $\mathbf{d}$, that represent the orientation of the propellers.

These are used in the optimization algorithm, as will be discussed next.

3.3 Controller Design

3.3.1 Multicopter Dynamics

In this section, we provide background information about multicopter dynamics. We first introduce the motor and propeller model then describe the multicopter dynamics based on Newton's law and Euler's equations.

Motor and Propeller

Our motor and propeller model is based on measuring multiple properties (such as thrust, torque, voltage, and current) then fitting them with analytical functions.

Thrust and torque We use u and τ to denote the magnitude of thrust and spinning torque induced by the propeller. The torque is known to be proportional to the thrust [87]:

$$\tau = \lambda u, \tag{3.1}$$

where λ is a constant ratio determined by the blade geometry, and is acquired by fitting the thrust and torque measurement.

Motor control The motor spinning rate is controlled by sending desired power-width modulation (PWM) signals to its electric speed controller (ESC). PWM signals control the power supplied to the motor, and therefore its spinning rate, which further influences the thrust and torque induced by the motor. We measure the mapping between PWM values, battery voltage and thrust, then use its inverse mapping to convert the output thrust from the controller to PWM values sent to each motor.

Power consumption When a motor loaded with a propeller is powered on, the spinning motor converts electronic power P_{ele}, the product of voltage and current I, into mechanic power which rotates the propeller, which then pushes surrounding air to generate thrusts u. We directly measure u-P_{ele} and u-I curves, which can be well approximated by power functions, and use them to guide our optimization on flight time and max amperage.

3.3.2 Equations of Motion

We use north-east-down (NED) coordinates as our world frame to determine the position and orientation of the copter. Our body frame is fixed at the center of the copter, and initially its three axis are parallel to the axis of world frame. To use propellers efficiently, we require all propellers to thrust upwards, not downwards. This can be guaranteed by matching the propeller type and motor spin direction.

The thrust and torque produced by the i-th motor are $u_i\mathbf{d}_i$ and $b_i\lambda_i u_i\mathbf{d}_i$, where $b_i \in \{-1, 1\}$ indicates counterclockwise or clockwise spinning direction and $\mathbf{d}_i$ is a unit vector representing the motor's orientation in the body frame. From Newton's second law and Euler's equation the dynamics are:

$$m\ddot{\mathbf{p}} = m\mathbf{g} + \mathbf{R}\sum_{i=1}^{n} u_i\mathbf{d}_i, \tag{3.2}$$

$$\mathbf{J}\dot{\omega} + \omega \times \mathbf{J}\omega = \sum_{i=1}^{n}(b_i\lambda_i u_i\mathbf{d}_i + \mathbf{r}_i \times u_i\mathbf{d}_i). \tag{3.3}$$

Since the net thrust and torque on the right side are linear on u_i, we introduce matrices $\mathbf{M}_f$ and $\mathbf{M}_t$ for compact representation:

$$m\ddot{\mathbf{p}} = m\mathbf{g} + \mathbf{R}\mathbf{M}_f\mathbf{u}, \tag{3.4}$$

$$\mathbf{J}\dot{\omega} + \omega \times \mathbf{J}\omega = \mathbf{M}_t\mathbf{u}. \tag{3.5}$$

Since the motor positions $\{\mathbf{r}_i\}$ can be computed from the shape parameter $\mathbf{s}$ by $\mathbf{r}_i = A_i\mathbf{s} + \mathbf{b}_i$, where all $\mathbf{A}_i$ and $\mathbf{b}_i$ are constant matrices and vectors from parametric shape representation, we will omit $\{\mathbf{r}_i\}$ and only use $\mathbf{s}$ in later sections.

3.3.3 Controller Design

In this section, we introduce the controller used in our multicopters. We start from reformulating the equations of motion into the state-space model, then we use its linear approximation at a fixed point to design an LQR controller.

State-Space Model

To design a controller, we rewrite the equations of motion into the following nonlinear form, known as the *state-space representation*:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \tag{3.6}$$

where $\mathbf{f}$ is a nonlinear function determined by multicopter dynamics. Intuitively, the state-space representation describes the fact that given the current state and the actuator output, we can predict how the state will change in the future.

Designing a controller for a nonlinear system is not an easy task, so we linearize the state-space model at a fixed point $(\mathbf{x}^*, \mathbf{u}^*)$ and find a controller for the linear model approximation. A fixed point satisfies $\mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) = \mathbf{0}$, which can be explained as a combination of state and thrust such that the copter can stay in this state forever, as long as the thrust does not change. Define $\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x}^*$, $\bar{\mathbf{u}} = \mathbf{u} - \mathbf{u}^*$, $\mathcal{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Big|_{\mathbf{x}^*, \mathbf{u}^*}$ and $\mathcal{B} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\Big|_{\mathbf{x}^*, \mathbf{u}^*}$, we can get the linear time-invariant (LTI) approximation of the state space model around $(\mathbf{x}^*, \mathbf{u}^*)$:

$$\dot{\bar{\mathbf{x}}} \approx \mathcal{A}\bar{\mathbf{x}} + \mathcal{B}\bar{\mathbf{u}}. \tag{3.7}$$

Linear Quadratic Regulator

The state-space model is different from the actual dynamics model because it is a linear approximation, and it assumes the state is in the neighborhood of the fixed point. As a result, we choose to use LQR because it is a robust controller with a phase margin of at least 60 degrees and an infinite gain margin [6], which means it can remain stable even if the dynamics model deviates a lot from our expectation. Given equation (3.7), LQR generates the control policy $\bar{\mathbf{u}} = -\mathcal{K}\bar{\mathbf{x}}$ by minimizing the cost function $\int_0^\infty (\bar{\mathbf{x}}^\top \mathcal{Q}\bar{\mathbf{x}} + \bar{\mathbf{u}}^\top \mathcal{R}\bar{\mathbf{u}})dt$, where $\mathcal{Q}$ and $\mathcal{R}$ are user-specified weight matrices which are usually positive diagonal. The first quadratic term tries to shrink $\bar{\mathbf{x}}$ to $\mathbf{0}$, so it penalizes the deviation from stable states. Similarly, the second term discourages actuators from saturation. The matrix $\mathcal{K}$ is found by solving the continuous-time algebraic riccati equation (CARE) once $\mathcal{A}$, $\mathcal{B}$, $\mathcal{Q}$ and $\mathcal{R}$ are given, which has been well-studied and implemented in many linear algebra packages [84, 103, 128].

Given multicopter dynamics, designing an LQR controller consists of two steps: selecting a fixed point for linearization, and solving CARE. For traditional quadcopters the first step is not a problem because it has a unique fixed point, i.e, each motor provides thrust equal to 1/4 of the gravity and the quadcopter stays completely level. However, we notice that general multicopter designs often have nonunique fixed points, and the performance of a flying multicopter is influenced by the choice of fixed points so arbitrarily picking a fixed point does not yield good results. To address this issue, we select the fixed point by including it in an optimization problem, which we will describe in the next section.

3.4 Co-Optimization

Optimizing a multicopter is a challenging problem because the performance of a multicopter relies on both its geometry and controller, which are usually coupled with each other. For example, to make sure a multicopter can hover, one needs to find reasonable motor positions and orientations, as well as good output thrusts suggested by the controller. Another challenge comes from the metrics that users want to optimize. For example, it could be a nonconvex function with multiple local minimals, or a nonsmooth function so a gradient-based solver is not applicable.

In this section we introduce our solution to the two challenges above. We provide an algorithm to decouple geometry and control variables during optimization. The geometry variables are motor spin directions $\{b_i\}$, motor orientations $\{\mathbf{d}_i\}$ and shape parameter $\mathbf{s}$. The control variables are the fixed point $(\mathbf{x}^*, \mathbf{u}^*)$ and the control matrix $\mathcal{K}$, as explained in Section 3.3. As $\mathbf{x}^*$ can be easily determined once $\mathbf{u}^*$ and geometry variables are given, and $\mathcal{K}$ can be determined after geometry and fixed point are known, the optimization algorithm only focuses on finding the optimal $\mathbf{u}^*$ and we leave the derivation of $\mathbf{x}^*$ in supplemental materials. For the metrics, we formulate an objective function including bi-convex user-defined metrics, and we demonstrate that many useful metrics can be represented as bi-convex functions.

The pseudocode for the complete optimization process is provided in Alg. 1. Our optimization starts from searching the discrete variables $\{b_i\}$ and selecting the combination that is most controllable, then in the main loop we solve three subproblems to optimize the control variable $\{\mathbf{u}^*\}$, shape parameter $\mathbf{s}$ and motor orientations $\{\mathbf{d}_i\}$ in every iteration. After that, we use the geometry variables to build the multicopter and control variables to implement the controller.

Algorithm 1: Optimization algorithm pseudocode.

```
Input  : Initial geometry variables, acquired from user design: ${\mathbf{d}_i}$, ${\mathbf{r}_i}$, $\mathbf{s}$
         User-specified LQR weight matrices $\mathcal{Q}$, $\mathcal{R}$
Output: Optimized geometry variables: ${b_i}$, ${\mathbf{d}_i}$, ${\mathbf{r}_i}$, $\mathbf{s}$;
         Optimized control variables(fixed point): $\mathbf{x}^*$, $\mathbf{u}^*$;
         control matrix $\mathcal{K}$
// Preprocessing
bestCondNumber $\leftarrow +\infty$
foreach $\mathbf{b} \in$ {all $2^n$ assignments to ${b_i}$ } do
    Compute $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$;
    if cond($\mathcal{C}$) $\leq$ bestCondNumber then
        bestCondNumber $=$ cond($\mathcal{C}$);
        ${b_i} \leftarrow \mathbf{b}$;
// Main loop
while not converge do
    Optimize $\mathbf{u}^*$ by solving a convex subproblem;
    Optimize $\mathbf{s}$ by solving a convex subproblem;
    Optimize ${\mathbf{d}_i}$ from QCQP relaxation;
    // Check controllability.
    Compute $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$;
    if $\mathcal{C}$ is singular then
        Revert all variables to their values in the last iteration;
        break;
// Postprocessing
Compute $\mathbf{x}^*$ from $\mathbf{u}^*$;
Compute $\mathcal{A}$, $\mathcal{B}$ from $(\mathbf{x}^*, \mathbf{u}^*)$;
Compute $\mathcal{K}$ from $\mathcal{A}$, $\mathcal{B}$, $\mathcal{Q}$ and $\mathcal{R}$;
```

3.4.1 Preprocessing

We first determine the spinning directions $\{b_i\}$ for each motor by checking the controllability matrix of the dynamics system. A system is controllable if for any initial state $\mathbf{x}_0$ and final state $\mathbf{x}_1$ there exists a control signal such that the system can steer from $\mathbf{x}_0$ to $\mathbf{x}_1$ in finite time [6]. An LTI system is controllable if the corresponding controllability matrix defined on the state-space model has full row rank. For equation (3.7), the controllability matrix $\mathcal{C}$ is defined as:

$$\mathcal{C} = \begin{bmatrix} \mathcal{B} & \mathcal{A}\mathcal{B} & \mathcal{A}^2\mathcal{B} & \cdots & \mathcal{A}^{11}\mathcal{B} \end{bmatrix}, \tag{3.8}$$

where the maximal exponent is 11 because $\bar{\mathbf{x}}$ consists of 12 variables. We check all 2^n possible combinations to determine the best $\{b_i\}$: for each assignment to $\{b_i\}$ we do linear approximation at its fixed point (if multiple fixed points exist we use the one with minimal L2 norm) then pick the assignment with the smallest condition number. Since n is up to 6 in our case, this straightforward method does not cause any performance issue.

Once $\{b_i\}$ is determined we keep them fixed during the optimization. Our algorithm in later sections checks at the end of each iteration whether the multicopter is still controllable, and it terminates with the best solution so far if controllability is violated. However, in our experiments this check rarely fails and the controllability property is preserved most of the time.

3.4.2 Problem Formulation

Here we give the formal definition of the optimization problem:

$$\begin{aligned} \min_{\mathbf{s},\mathbf{d}_i,\mathbf{u}^*} \quad & E(\mathbf{u}^*, \mathbf{s}) + \eta\|\mathbf{M}_f\mathbf{u}^* + m\mathbf{g}\|_2^2 + \mu\|\mathbf{M}_t\mathbf{u}^*\|_2^2. && (3.9) \\ s.t. \quad & \mathbf{A}^{\text{ineq}}\mathbf{s} \le \mathbf{b}^{\text{ineq}}, && (3.10) \\ & \mathbf{d}_i^\top\mathbf{d}_i = 1, && (3.11) \\ & \mathbf{0} \le \mathbf{u}^* \le \mathbf{u}^{\max}. && (3.12) \end{aligned}$$

where E is the user-selected metrics, which we will describe in the next section. We require that E rely on the shape parameters $\mathbf{s}$ and thrust $\mathbf{u}^*$ only, and be bi-convex. The second and third terms model soft constraints required by the fixed point with user specified weights η and μ: By definition, at a fixed point the net thrust should try to balance the gravity, and the net torque should be zero. Compared with the equations of motion the rotational matrix $\mathbf{R}$ is removed to reduce the complexity, and to indicate that it is preferable for the copter to maintain its original attitude as much as possible, i.e., $\mathbf{R}$ is equal to an identity matrix, which makes the copter easier to take off. Note that $\mathbf{M}_f$ relies on $\mathbf{d}_i$ and $\mathbf{M}_t$ depends on $\mathbf{d}_i$ and $\mathbf{s}$.

The first constraint gives the feasible set of the shape parameter. Constraints on $\mathbf{d}_i$ require that each $\mathbf{d}_i$ should be a unit vector. The last constraint requires that no motor saturation should occur.

3.4.3 Metrics

Although we limit the energy function to be bi-convex, we demonstrate that a lot of metrics can fit into this representation either directly or after reasonable reformula-

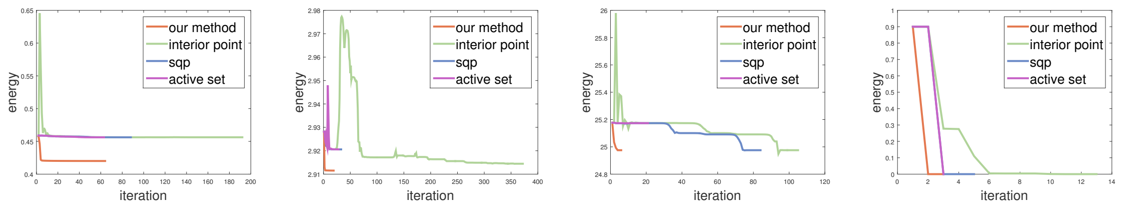


Figure 3-3: Comparing our method in multiple copter examples with interior point, sequential quadratic programming and active set methods, all implemented in MATLAB's `fmincon` command. All examples have $\eta = 0.3$ and $\mu = 0.7$, and all methods share the same initial guess and termination conditions. The horizontal axis shows the iterations (note that it does not reflect the true running time as the time an iteration takes varies in different methods) and the vertical axis is the value of the objective function in Eqn. (3.9) after each iteration. Left: pentacopter with payload metric. Middle left: bunny with mixed metrics of payload and amperage. Middle right: pentacopter with mixed metrics of max amperage and cost. Right: quadcopter with size metric.

tion. Here we list some:

Payload We define the payload to be the maximal weight the copter can take at its mass center while hovering. Maximizing the payload directly results in a non-convex formulation with all variables closely coupled with each other. Instead, we minimize the following indirect metric:

$$E_{\text{payload}} = \max\left(\frac{u_1^*}{u_1^{\max}}, \frac{u_2^*}{u_2^{\max}}, \cdots, \frac{u_n^*}{u_n^{\max}}\right), \tag{3.13}$$

where E can be explained as searching all the motors and finding the one that is most likely to become saturated. If a copter can hover with thrust equal to $\mathbf{u}$, scaling $\mathbf{u}$ uniformly allows a copter with same geometry but heavier weight to stay in the air. As a result, smaller E indicates possibly larger payload.

Max amperage The relation between thrust and current supplied to the motor can be well approximated by a power function $I = au^{\alpha}$. Since the power module unit in hardware platform distributes the current from the battery to all motors, for safety reasons we are interested in minimizing the total current so that we do not exceed the maximal amperage of the power module cable:

$$E_{\text{amp\_sum}} = \sum_{i}^{n} a_i u_i^{*\alpha_i}, \tag{3.14}$$

where $a_i u_i^{*\alpha_i}$ represents the current supplied to i-th motor. Alternatively, we can minimize the max current supplied to a single motor so that it does not exceed the maximum amperage of the electric speed controller of each motor:

$$E_{\text{amp\_max}} = \max(a_1 u_1^{*\alpha_1}, a_2 u_2^{*\alpha_2}, \cdots, a_n u_n^{*\alpha_n}). \tag{3.15}$$

Size and flight time To minimize the copter size, we choose to minimize the total length of rods used in the copter, which is linear on the shape parameter $\mathbf{s}$:

$$E_{\text{size}} = \mathbf{c}_{\text{rod}}^{\top}\mathbf{s} + d_{\text{rod}}. \tag{3.16}$$

We can also relate the copter size to its flight time. From our measurement we notice that larger propellers are more efficient than smaller ones with the motors we use. As a result, we can increase the flight time by maximizing the copter size so that it has enough space to install larger propellers:

$$E_{\text{time}} = -E_{\text{size}} = -(\mathbf{c}_{\text{rod}}^{\top}\mathbf{s} + d_{\text{rod}}). \tag{3.17}$$

The negative sign comes from the fact that we minimize the objective function. It should be pointed out that this energy function is based on the observation that larger propellers are more efficient than smaller ones with our motor, which is not always true for all combinations of motors and propellers. Careful measurement needs to be taken before applying this metric.

Cost For a multicopter without a free-form body frame the cost is fully determined by the shape parameter $\mathbf{s}$ in a linear form:

$$E_{\text{cost}} = \mathbf{c}_{\text{cost}}^{\top}\mathbf{s} + d_{\text{cost}}. \tag{3.18}$$

where $\mathbf{c}_{\text{cost}}$ represents the linear cost like carbon fiber rods, and d_{cost} is the constant cost from components such as battery, controlling board and motors.

Mass For a multicopter without a free-form body frame the mass is linear on the shape parameter $\mathbf{s}$

$$E_{\text{mass}} = \mathbf{c}_{\text{mass}}^{\top}\mathbf{s} + d_{\text{mass}}. \tag{3.19}$$

When applying this metric, m in the objective function is replaced with $m(\mathbf{s}) = E_{\text{mass}}$. However, this modification won't break our proposed algorithm as the mass is linear on the shape parameters, so it should not break the convexity of the subproblem.

Multi-objective metric Since a non-negative weighted sum preserves convexity, any nonnegative weighted combination of the metrics above meets our bi-convexity requirement. For examples, users may choose to define a metric mixed with both payload and mass, and use weights to express the tradeoff between them. Also note that although all metrics above are defined solely on $\mathbf{u}^*$ or $\mathbf{s}$, a mixed metric can include both of them.

3.4.4 Algorithm

Directly optimizing the objective function proposed in the previous section is challenging due to the fact that the product between $\mathbf{M}_f$, $\mathbf{M}_t$, and $\mathbf{u}^*$ couples all variables together, and it may contain non-smooth energy functions like the payload or max amperage. However, if we focus on $\mathbf{u}^*$ or $\mathbf{s}$ only, it reduces to a simple convex problem. This motivates us to propose our algorithm which alternatively optimizes the thrust, shape parameters and motor directions, as described in Alg. 1. It terminates when the energy converges within a given threshold (1e-6 in our setting) or it breaks the controllability. With the assumption that user metrics are bi-convex functions, the problem is convex when restricted to optimizing either $\mathbf{u}^*$ or $\mathbf{s}$, which can be efficiently solved in CVX [60, 59], a package for specifying and solving convex programs.

Optimizing $\{\mathbf{d}_i\}$ is more subtle and bears some discussion. The objective function is quadratic and convex on $\{\mathbf{d}_i\}$ due to the fact that both $\mathbf{M}_f$ and $\mathbf{M}_t$ are linear on $\{\mathbf{d}_i\}$, but the unit-length constraints on $\{\mathbf{d}_i\}$ breaks the convexity. Because of its quadratic form in the objective function, we choose Sequential Quadratic Programming (SQP) [110] to solve $\{\mathbf{d}_i\}$. The initial guess is acquired from relaxing the unit length constraints to $\mathbf{d}_i^\top \mathbf{d}_i \leq 1$, and then solving the convex Quadratic Constrained Quadratic Programming (QCQP) problem [23].

We demonstrate that our algorithm is more suitable than three other general solvers for our optimization problem in Fig. 3-3. In most of the time our algorithm manages to find a better solution, but with the cost of longer time. However, this is not a big issue as our solver usually terminates within a few seconds.

In the first two examples, we run our method on two different multicopter designs with single and mixed metrics. For both examples our algorithm quickly finds a better solution after the first few iterations, while the other solvers either fail to make progress or get trapped into a worse local minimal solution. There exist some cases where our algorithm ends up with similar optimal values, in which case it is still acceptable to call our solver as the system is not sensitive to its running time.

Finally, as a sanity check we provide an example where the global minimizer is known. In this example a standard quadcopter is optimized to have minimal size. Without other constraints the copter shrinks to a single point and the energy function becomes zero. In this case all solvers agree on the global minimizer in the end.

3.5 Simulation

We provide a real-time physics simulator to help users verify the shape and controller design suggested by the optimization. Users can interactively change the input from a virtual RC transmitter and see the flight performance of the copter. Random noises are added to the sensor data to simulate real world environment. If the optimized shape and controller are not satisfactory, users can either manually tweak control parameters in simulation, or go back to the interactive design tool to change the shape representation.

3.6 Fabrication

Once the shape parameter and motor orientations are determined, we generate a fabrication plan by computing rod lengths, motor angles, and geometry meshes if it

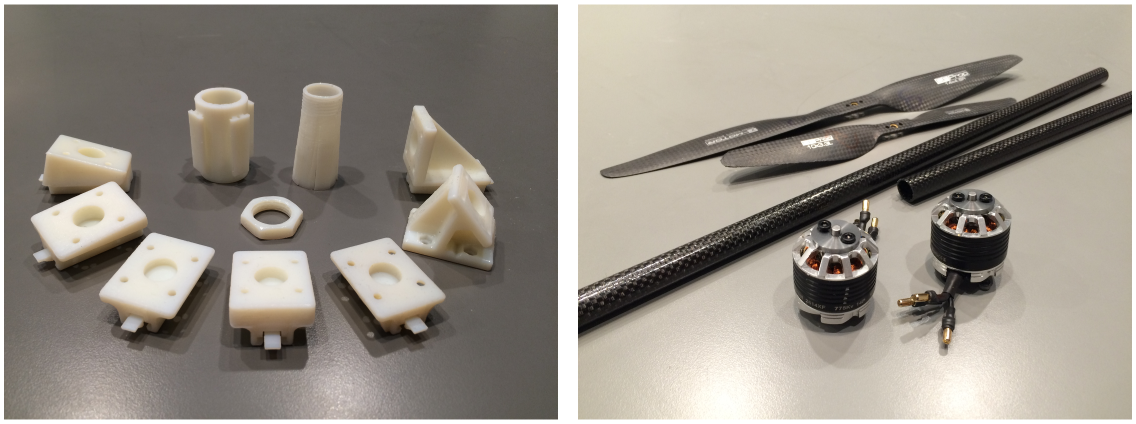


Figure 3-4: Some multicopter components used in fabrication. Left: 3D printed connectors that support various motor orientations. Right: propellers, carbon fiber rods and motors.

contains a free-form body frame (Fig. 3-4). We use 3D printing to fabricate connectors and the body frame. Specifically, we design parametric compound angle clip pieces so that the connectors can support tilted motors.

Our hardware platform consists of a ground station laptop and a multicopter flight controller. The ground station subscribes the Vicon motion tracking system and sends out real-time position and orientation data to the copter. The flight controller runs our modified version of the open source software ArduPilot [5] on a Pixhawk [106] flight computer hardware.

3.7 Results

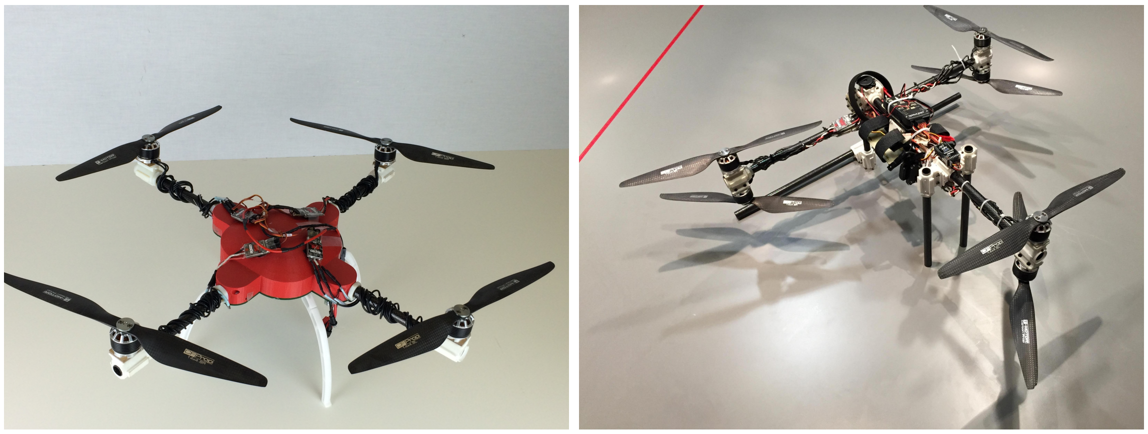


Figure 3-5: Classic designs. Left: a quadcopter with a free-form body frame. Right: a hexacopter with coaxial propellers.

In this section we show multiple examples to demonstrate the effectiveness of our design system, physical simulator, control loop and optimization method. We start from two classic multicopters: an X-frame quadcopter and a Y6 hexacopter. We pro-

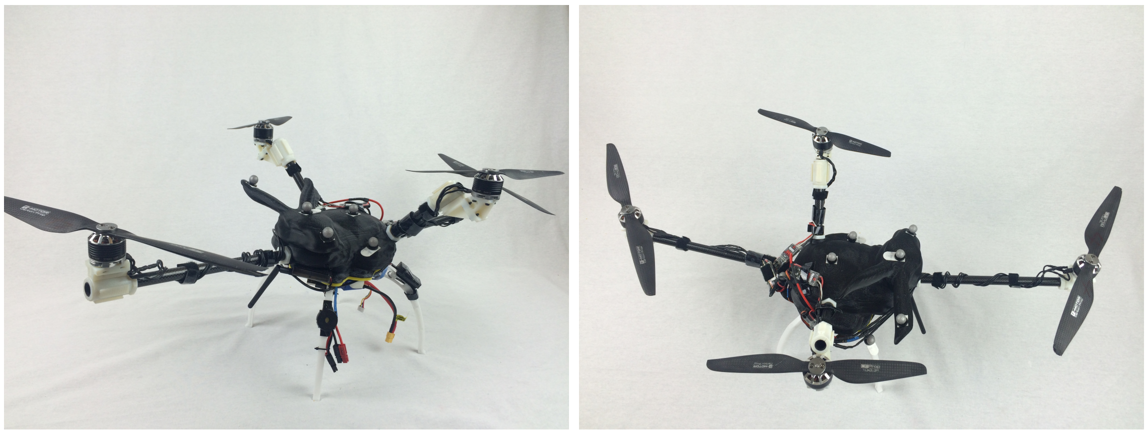


Figure 3-6: Bunny copter (left) and its top-down view (right). Note that the positions of motors are not symmetric, their propeller sizes are different and are at various heights.

vide a bunny example to show the expressiveness of our design tool. A pentacopter demonstrates the correctness of our controller and its ability to handle nonstandard copters with odd number of motors. A more challenging pentacopter with tilted motors, which is optimized for the payload metric, shows the efficacy of our optimization method. Finally, we provide a rectangular quadcopter optimized for longer flight time and compare it with a standard quadcopter.

Quadcopter We design a simple quadcopter with a 3D-printed red body frame, shown in Fig. 3-5. The red frame is a parametric shape, so users can change its size by specifying different shape parameters in the interactive design tool. For simplicity, the default quadcopter PID controller in ArduPilot is used here, so Vicon is not needed and no modification to ArduPilot firmware is required.

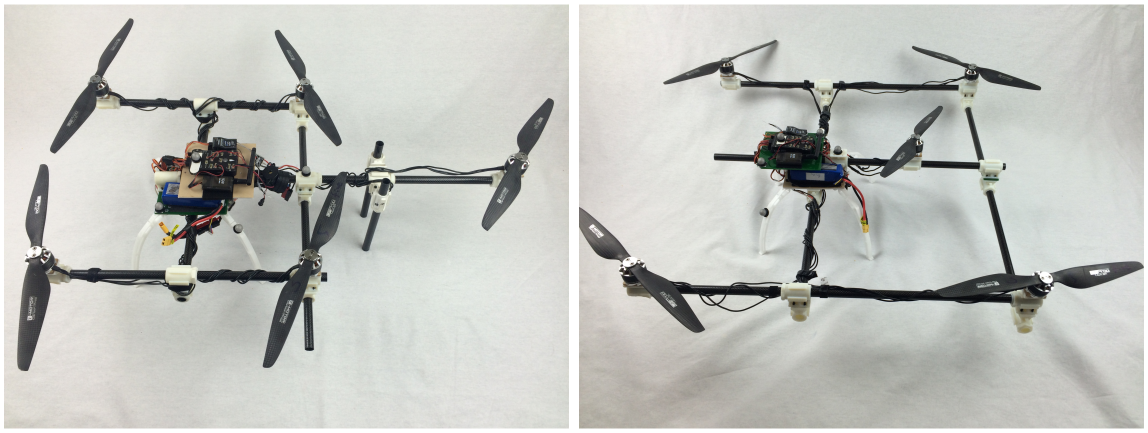


Figure 3-7: Pentacopter pairs. Left: original pentacopter design. Right: optimized pentacopter for larger payload.

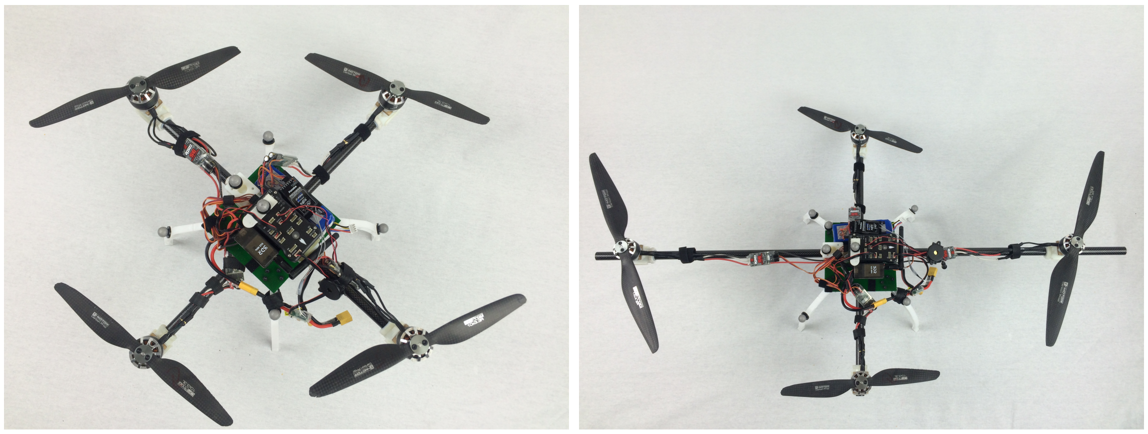


Figure 3-8: Optimizing a quadcopter with flight time metric and geometry constraints. Left: a standard quadcopter. Right: optimized rectangular quadcopter.

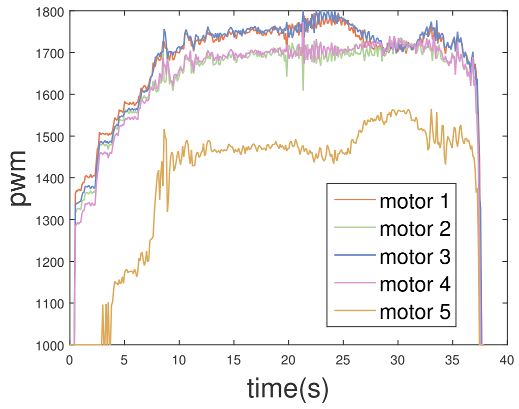
Hexacopter We design and fabricate a classic Y6 hexacopter with three pairs of coaxial propellers (Fig. 3-5). As in the previous example, the default Y6 PID controller in ArduPilot is applied and no additional hardware is needed. The hexacopter can change its heading, stabilize itself and fly to a target during the flight.

Bunny A bunny copter is designed and fabricated using our system (Fig. 3-6). The bunny copter is challenging to fly as the four propellers have different sizes, their positions are not symmetric and they are placed at different heights. Based on its dynamics we compute an LQR controller to control its position in the air. The bunny copter can take off, hover, fly to a target and land.

Unoptimized pentacopter Fig. 3-7 shows a multicopter with five rotors all pointing upright. Flying a multicopter with odd number of rotors, even if it is symmetric, is challenging because there is no straightforward way to distribute thrust so that all motor torques can be balanced easily. However, with the LQR controller suggested by our system this pentacopter can reliably take off, land, hover, and carry over 1kg payload to the destination.

Fig. 3-9 shows the real-time output of all the 5 motors when this pentacopter carries maximal payload from one place to another. Although by default the motor does not saturate until PWM reaches 2000, in this example and its optimized counterpart we clamp PWM at 1800 for safety reasons.

Pentacopter optimized for payload Given the initial unoptimized pentacopter, our optimization improves its payload by changing its geometry and tilting motors to balance thrust from all motors. Fig. 3-9 shows the PWM values from all 5 motors when the optimized pentacopter carries its maximal payload, and Table 3.1 compares the specifications of two pentacopters. Our optimization result predicts the new pentacopter is able to take off with a 15.8% increase in the overall weight. Note that in theory the maximal possible increase in the overall weight is less than 25%



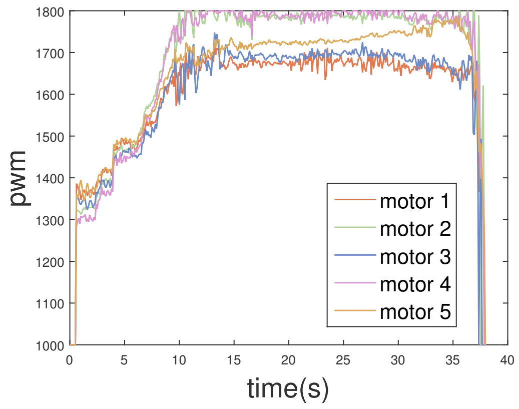


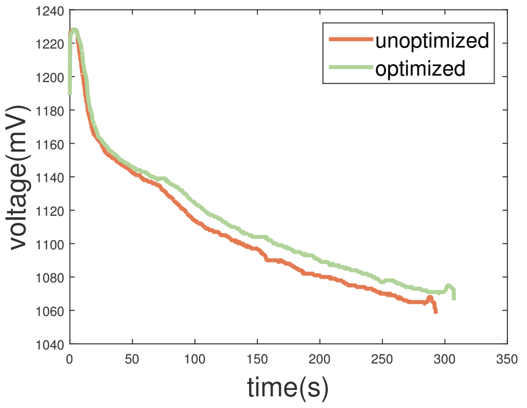
Figure 3-9: Left: motor outputs of the unoptimized pentacopter with 1047g payload. Motor 1 and 3 reach saturation point (PWM=1800) at 23s. Increasing the payload will cause them to saturate constantly and therefore fail to balance the torques from other motors. Note that motor 5 is not fully exploited in this copter. Right: motor outputs of the optimized pentacopter with 1392g payload. Motor 2 and 4 reach saturation during the flight. Compared with the unoptimized pentacopter, all five motors are now well balanced, making it possible to take over 30% more payload.

Table 3.1: Pentacopter specifications. Motor angle is the angle between motor orientation and up direction.

	Unoptimized	Optimized
Size (mm×mm×mm)	750×420×210	650×670×210
Weight (g)	2322	2353
Max payload (g)	1047	1392
Max overall weight (g)	3369	3745
Max motor angle (degree)	0	10.6

because even the unoptimized pentacopter outperforms a quadcopter whose maximal overall weight is $4u^{\max}$, and the maximal possible overall weight of a pentacopter is not greater than $5u^{\max}$. Table 3.1 shows that we get 11.1% actual gain in our experiments. The main reason for this loss is that we did not take into account the interference between propellers, which we leave as future work.

Quadcopter optimized for flight time Fig. 3-8 shows a standard quadcopter and its optimized version which has longer flight time. Our optimization tries to increase the total length of rods so that it makes room for larger propellers. In this example we add an upper bound constraint on the copter width so it only scales in the other direction, allowing us to replace the propellers in the longer rod with larger ones. This geometry constraint is useful when a quadcopter is designed to fly into a tunnel. Both copters are controlled by LQR controllers computed with our system. In our experiments we let both copters hover for 5 minutes and record the battery



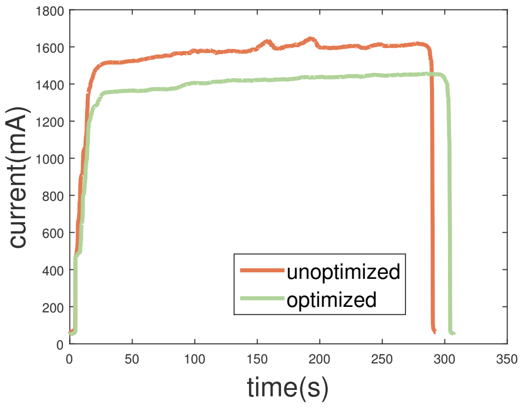


Figure 3-10: Battery change when a quadcopter hovers. Left: battery voltage. Given the same amount of time the optimized quadcopter ends up having a larger voltage; Right: battery current. In steady state the optimized copter requires less current.

voltage and current, shown in Fig. 3-10.

3.8 Conclusion

In this work, we propose a new pipeline for users to efficiently design, optimize, and fabricate multicopters. Users can easily design a multicopter by interactively assembling components in a user interface. We propose a new optimization algorithm that can jointly optimize the geometry and controller to improve the performance of a given multicopter design under different metrics, such as payload and battery usage, and can be further verified in a real-time simulator. We demonstrate the ability of our system by designing, fabricating, and flying multicopters with nonstandard designs including asymmetric motor configurations and free-form body frames.

Although not formally defined, the key ingredient in our computational pipeline is to compute the gradients for the shape and controller parameters, which allows us to run gradient-based continuous optimization algorithms. In the next chapter, we will formalize this idea as a differentiable simulator and elaborate on its technical details.

One limitation in our pipeline is that the system responds passively to user inputs for design, optimization and simulation. A potential extension in the future is to have a system that can actively give design suggestions in this case, for example by suggesting to place additional motors and propellers, so the whole design process can be accelerated. In particular, our system fails when the initial geometric design is uncontrollable. For example, a quadcopter initialized with 4 rotors in a line is obviously not fully controllable. In this case, our algorithm gets trapped in assigning spinning directions (Sec. 3.4), and therefore fails to find a controllable solution, which clearly exists for a quadcopter.

In terms of design, our assembly based approach depends on the library of parts and is therefore limited by its size. In the future it would be nice to define ways to easily grow the database. In additional, the parametrization is constrained to be linear which restricts geometry variability.

In terms of the optimization, the metrics we propose are limited to functions defined on the fixed points, i.e., the steady states of the copter. While this simplifies the optimization by decoupling the geometry and control variables, it would be useful to extend the optimization so that dynamic metrics can be included and optimized, for example the responsive time to a control signal, or the maneuverability of the copter.

Another limitation is that our real-time physics simulation does not model aerodynamic effects. While a rigid-body simulation provides reasonable results for cases with slow velocity, aerodynamic effects such as interferences between propellers and ground effects need to be modeled to simulate a high speed copter correctly.

Chapter 4

Differentiable Physics Simulation

In the last chapter, we have presented a computational pipeline for designing customized multicopters. The computational method manages to discover novel pentacopter designs that easily outperform a design suggested by experienced mechanical engineers by a large margin. The most critical component in the proposed pipeline is computing gradients for continuous shape and control parameters. In this chapter, we will re-introduce this idea as differentiable physics simulation and explain how to equip a physics simulator with gradients by presenting two advanced differentiable simulators for soft bodies [44] and fluids [45]. We show in Fig. 4-1 the position of this chapter in the entire computational robot design pipeline.

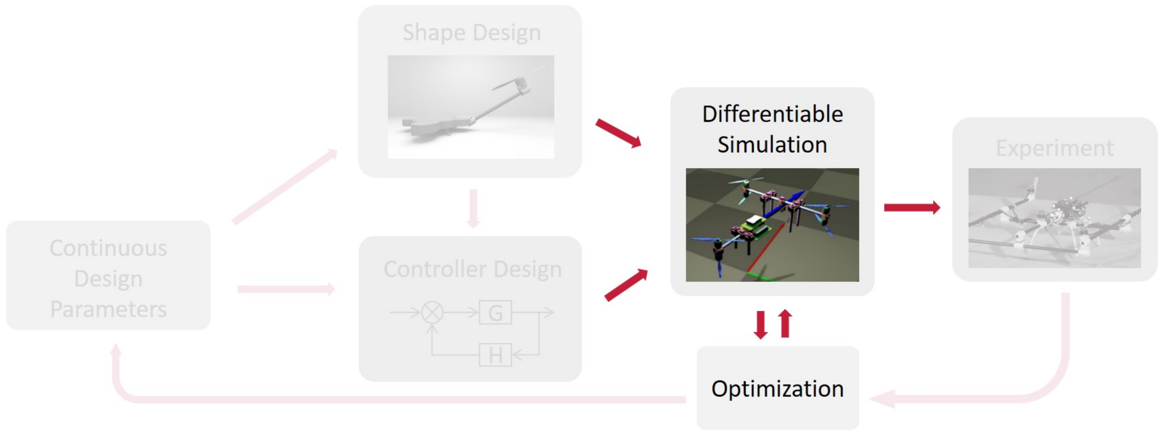


Figure 4-1: This chapter presents in depth the technical details of two differentiable simulators we develop in this thesis. As shown in the pipeline, a differentiable simulator plays a central role that connects the upstream and downstream applications.

4.1 Differentiable Soft Body Simulation

4.1.1 Motivation

The recent surge of differentiable physics witnessed the emergence of differentiable simulators as well as their success in various inverse problems that have simulation inside an optimization loop. With additional knowledge of gradients, a differentiable simulator provides more guidance on the evolution of a physics system. This extra information, when properly combined with mature gradient-based optimization techniques, facilitates the quantitative study of various downstream applications, e.g., system identification or parameter estimation, motion planning, controller design and optimization, and inverse design problems.

In this section, we focus on the problem of developing a differentiable simulator for soft-body dynamics. Despite its potential in many applications, research on differentiable soft-body simulators is still in its infancy due to the large number of degrees of freedom (DoFs) in soft-body dynamics. One learning-based approach is to approximate the true, soft-body dynamics by way of a neural network for fast, automatic differentiation [92, 121]. For these methods, the simulation process is no longer physics-based, but purely based on a neural network, which might lead to physically implausible and uninterpretable results and typically do not generalize well.

Another line of research, which is more physics-based, is to differentiate the governing equations of soft-body dynamics directly [69, 56, 68, 62]. We classify these simulators into *explicit* and *implicit* simulators based on their time-stepping schemes. Explicit differentiable simulators implement explicit time integration in forward simulation and directly apply the chain rule to derive any gradients involved. While explicit differentiable simulation is fast to compute and straightforward to implement, its explicit nature requires a tiny time step to avoid numerical instability. Moreover, when deriving the gradients, an output value (typically a reward or an error metric) needs to be backpropagated through all time steps. Such a process requires the state at every time step to be stored in memory regardless of the time-stepping scheme. Therefore, explicit differentiable simulators typically consume orders of magnitude more memory than their implicit counterparts and sophisticated schemes like checkpoints are needed to alleviate the memory issue [69].

Unlike explicit differentiable simulators, implicit simulation enables a much larger time step. It is more robust numerically and much more memory-efficient during backpropagation. However, an implicit differentiable simulator typically implements Newton's method in forward simulation and the adjoint method during backpropagation [62, 56], both of which require the expensive linearization of the soft-body dynamics. Even though techniques for expediting the *forward* soft-body simulation with an implicit time-stepping scheme have been developed extensively, the *backpropagation* process remains a bottleneck for downstream applications in inverse problems.

In this section, we present DiffPD, a efficient differentiable soft-body simulator that implements the finite element method (FEM) and an implicit time-stepping scheme with certain assumptions on the material and contact model. We draw inspiration from Projective Dynamics (PD) [22], a fast and stable algorithm that can be used for solving implicit time integration with FEM when the elastic energy of the material model has a specific quadratic form. The key observation we share with PD is that the computation bottleneck in both forward simulation and backpropagation is due to the nonlinearity of soft-body dynamics. By decoupling nonlinearity in the system dynamics, PD proposes a global-local solver where the global step solves a prefactorized linear system of equations and the local step resolves the nonlinearities in the physics and can be massively parallelized. Previous work in PD has demonstrated its efficacy in forward simulation and has reported significant speedup over the classical Newton's method. Our core contribution is to establish that with proper linear algebraic reformulation, the same idea of nonlinearity decomposition from PD can be fully extended to backpropagation as well.

To support differentiable contact handling, we revisit contact models used in previous PD papers, many of which choose to implement a soft contact force based on a fictitious collision energy [153, 154, 40, 39, 96, 89, 22]. DiffPD naturally supports such energy-based contact and friction models as they can be seamlessly integrated into the PD framework. One notable exception is Ly et al. [97], which solves dry frictional contact in the standard PD framework. We have also explored the possibility of making the dry frictional contact model differentiable in DiffPD. Using the fact that contact vertices must be on the soft body's surface, which typically have much fewer DoFs than the interior vertices, we present a novel solution combining Cholesky factorization and low-rank matrix update to supporting differentiable *static* friction and non-penetration contact.

We demonstrate the efficacy of DiffPD in various 3D applications with up to nearly $30,000$ DoFs. These applications include system identification, initial state optimization, motion planning, end-to-end closed-loop control optimization, and estimation of contact and friction properties from a real-world experiment. We compare DiffPD to both explicit and implicit differentiable FEM simulations and observe DiffPD's forward and backward calculation is 4-19 times faster than Newton's method when assumptions in PD hold. Furthermore, we embed DiffPD as a differentiable layer in a deep learning pipeline for training closed-loop neural network controllers for soft robots and report a speedup of 9-11 times in wall-clock time compared to deep reinforcement learning (RL) algorithms. Finally, we show a reality-to-simulation (real-to-sim) application that uses our differentiable simulator to reconstruct a collision event between two tennis balls from a video input, which we hope can inspire follow-up work to solve simulation-to-reality (sim-to-real) problems in the future.

To summarize, this section contributes the following:

- A PD-based differentiable soft-body simulator that is significantly faster than differentiable simulators using the standard Newton's method;
- Differentiable collision handling for penalty-based contact and friction forces or complementarity-based non-penetration contact and static friction;
- Demonstrations of the efficacy of our method on a wide range of applications, including system identification, inverse design problems, motion planning, robotics control, and a real-to-sim experiment, using material models compatible with PD and the simplified contact model stated above.

4.1.2 Governing Equations

In this section, we review the basic concepts of the implicit time-stepping scheme and PD. Let n be the number of 3D nodes in a deformable body after FEM-discretization.

After time discretization, we use $\mathbf{x}_i \in \mathcal{R}^{3n}$ and $\mathbf{v}_i \in \mathcal{R}^{3n}$ to indicate the nodal positions and velocities at the i-th time step.

Implicit time integration In this work, we focus on the implicit time integration:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_{i+1}, \tag{4.1}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + h\mathbf{M}^{-1}[\mathbf{f}_{\text{int}}(\mathbf{x}_{i+1}) + \mathbf{f}_{\text{ext}}], \tag{4.2}$$

where h is the time step, $\mathbf{M} \in \mathcal{R}^{3n\times 3n}$ is a lumped mass matrix, and $\mathbf{f}_{\text{int}}$ and $\mathbf{f}_{\text{ext}}$ are the sum of the internal and external forces, respectively. Substituting $\mathbf{v}_{i+1}$ in $\mathbf{x}_{i+1}$ gives the following nonlinear system of equations:

$$\frac{1}{h^2}\mathbf{M}(\mathbf{x}_{i+1} - \mathbf{y}_i) - \mathbf{f}_{\text{int}}(\mathbf{x}_{i+1}) = \mathbf{0}, \tag{4.3}$$

where $\mathbf{y}_i = \mathbf{x}_i + h\mathbf{v}_i + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$ is evaluated at the beginning of each time step. We drop the indices from $\mathbf{x}$ and $\mathbf{y}$ for simplicity:

$$\frac{1}{h^2}\mathbf{M}(\mathbf{x} - \mathbf{y}) - \mathbf{f}_{\text{int}}(\mathbf{x}) = \mathbf{0}. \tag{4.4}$$

At each time step, our goal is to find $\mathbf{x}$ satisfying the equation above with the given $\mathbf{y}$.

As pointed out by Stuart et al. [139] and Martin et al. [101], solving $\mathbf{x}$ from Eqn. (4.4) is equivalent to finding the critical point of the following objective g:

$$g(\mathbf{x}) = \frac{1}{2h^2}(\mathbf{x} - \mathbf{y})^\top\mathbf{M}(\mathbf{x} - \mathbf{y}) + E(\mathbf{x}), \tag{4.5}$$

where E is the potential energy that induces the internal force: $\mathbf{f}_{\text{int}} = -\nabla E$. It is easy to check the left-hand side of Eqn. (4.4) is ∇g.

Eqn. (4.4) is typically solved with Newton's method, which iteratively solves a series of linear systems of equations. Consider the k-th iteration in Newton's method with $\mathbf{x}^k$ being the guess on $\mathbf{x}$ so far. Newton's method computes the next guess on $\mathbf{x}$ as follows:

$$\mathbf{0} = \nabla g(\mathbf{x}) = \nabla g(\mathbf{x}^k + \Delta\mathbf{x}) \approx \nabla g(\mathbf{x}^k) + \nabla^2 g(\mathbf{x}^k)\Delta\mathbf{x}, \tag{4.6}$$

$$\nabla^2 g(\mathbf{x}^k)\Delta\mathbf{x} \approx -\nabla g(\mathbf{x}^k). \tag{4.7}$$

Therefore, one can let $\Delta\mathbf{x} = -[\nabla^2 g(\mathbf{x}^k)]^{-1}\nabla g(\mathbf{x}^k)$ and update their guess on $\mathbf{x}$ at the next iteration by $\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}$. In practice, Newton's method typically employs definiteness fixes or line searches when $\nabla^2 g$ is indefinite [110]. For large-scale problems, solving Eqn. (4.7) at each $\mathbf{x}^k$ requires expensive linearization and matrix factorization, which becomes the time bottleneck.

Backpropagation with implicit time integration We sketch the main idea of backpropagation with a loss function L defined on $\mathbf{x}$ and explain how we can compute $\frac{\partial L}{\partial \mathbf{y}}$ from $\frac{\partial L}{\partial \mathbf{x}}$. Backpropagating through multiple time steps can be done by backpropagating through every single pair of $(\mathbf{x}, \mathbf{y})$ from each time step repeatedly. As $\mathbf{x}$ and $\mathbf{y}$ are implicitly constrained by $\nabla g(\mathbf{x}) = \mathbf{0}$, we can differentiate it with respect to $\mathbf{y}$ and obtain the following equation:

$$\nabla^2 g(\mathbf{x}) \frac{\partial \mathbf{x}}{\partial \mathbf{y}} - \frac{1}{h^2}\mathbf{M} = \mathbf{0}. \tag{4.8}$$

We can solve $\frac{\partial \mathbf{x}}{\partial \mathbf{y}}$ from it and use the chain rule to obtain the following (assuming both $\frac{\partial L}{\partial \mathbf{x}}$ and $\frac{\partial L}{\partial \mathbf{y}}$ are row vectors):

$$\frac{\partial L}{\partial \mathbf{y}} = \frac{\partial L}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{y}} = \frac{1}{h^2}\underbrace{\frac{\partial L}{\partial \mathbf{x}}[\nabla^2 g(\mathbf{x})]^{-1}}_{\mathbf{z}^\top}\mathbf{M}. \tag{4.9}$$

Note that the inverse of $\nabla^2 g(\mathbf{x})$ is intentionally regrouped with $\frac{\partial L}{\partial \mathbf{x}}$ to avoid the expensive $[\nabla^2 g(\mathbf{x})]^{-1}\mathbf{M}$. The adjoint vector $\mathbf{z}$ can be solved from the following linear system of equations:

$$\nabla^2 g(\mathbf{x})\mathbf{z} = (\frac{\partial L}{\partial \mathbf{x}})^\top. \tag{4.10}$$

Note that we drop the transpose of $\nabla^2 g(\mathbf{x})$ because it is symmetric.

Putting them together, we have shown that backpropagation within one time step can be done by Eqns. (4.9) and (4.10). It is now clear that $\nabla^2 g(\mathbf{x})$ plays a crucial role in both forward simulation and backpropagation, and we write its definition explicitly below:

$$\nabla^2 g(\mathbf{x}) = \frac{1}{h^2}\mathbf{M} + \nabla^2 E(\mathbf{x}). \tag{4.11}$$

Similar to Newton's method in forward simulation, a direct implementation of Eqn. (4.10) is computationally expensive because $\nabla^2 g(\mathbf{x})$ needs to be reconstructed and refactorized at every time step. This motivates us to propose the novel PD-based backpropagation method in Sec. 4.1.3.

Projective Dynamics PD considers a specific family of quadratic potential energies that decouple the nonlinearity in material models [22]. Specifically, PD assumes the energy E is the sum of quadratic energies taking the following form:

$$E_c(\mathbf{x}) = \min_{\mathbf{p}_c \in \mathcal{M}_c} \underbrace{\frac{w_c}{2}\|\mathbf{G}_c\mathbf{x} - \mathbf{p}_c\|_2^2}_{\tilde{E}_c(\mathbf{x},\mathbf{p}_c)}, \tag{4.12}$$

$$E(\mathbf{x}) = \sum_c E_c(\mathbf{x}), \tag{4.13}$$

where $\mathbf{G}_c$ is a discrete differential operator in the form of a constant sparse matrix, w_c is a scalar that determines the stiffness of the energy, and $\mathcal{M}_c$ is a constraint manifold. For example, if one wants to formulate a volume-preserving elastic energy, $\mathcal{M}_c$ can

Algorithm 2: PD forward simulation in one time step

```
Input: $\mathbf{y}$;
Output: $\mathbf{x}$ that satisfies Eqn. (4.4);
Initialize $\mathbf{x} = \mathbf{y}$;
while $\mathbf{x}$ not converged do
    $\mathbf{p}_c = \arg\min_{\mathbf{p}_c \in \mathcal{M}_c} \tilde{E}_c(\mathbf{x}, \mathbf{p}_c)$;  // Local step;
    $\mathbf{b} = \frac{1}{h^2}\mathbf{M}\mathbf{y} + \sum_c w_c \mathbf{G}_c^\top \mathbf{p}_c$;
    $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$;  // Global step;
```

be the set of all 3×3 matrices whose determinant is 1. E_c is defined as the distance from $\mathbf{G}_c\mathbf{x}$ to $\mathcal{M}_c$. Following the prevalent practice in previous work [22, 96, 107], we assume E_c is defined on each finite element with $\mathbf{G}_c$ mapping $\mathbf{x}$ to the local deformation gradients $\mathbf{F}$ [130].

With the definition of E at hand, PD obtains the critical point of g by alternating between a local step and a global step, which essentially minimizes the following surrogate objective $\tilde{g}(\mathbf{x}, \mathbf{p})$:

$$\tilde{g}(\mathbf{x}, \mathbf{p}) = \frac{1}{2h^2}(\mathbf{x} - \mathbf{y})^\top \mathbf{M}(\mathbf{x} - \mathbf{y}) + \sum_c \tilde{E}_c(\mathbf{x}, \mathbf{p}_c), \tag{4.14}$$

where $\mathbf{p}$ stacks up all $\mathbf{p}_c$ from each E_c. The local and global steps in PD can be interpreted as running coordinate descent optimization on $\tilde{g}$. The local step fixes the current $\mathbf{x}$ and projects $\mathbf{G}_c\mathbf{x}$ onto $\mathcal{M}_c$ to obtain $\mathbf{p}_c$ in each E_c, which can be massively parallelizable across all E_c. The global step fixes $\mathbf{p}$ and minimize $\tilde{g}$ over $\mathbf{x}$, which turns out to be a quadratic function with an analytical solution solved from the following linear system of equations:

$$\underbrace{(\frac{1}{h^2}\mathbf{M} + \sum_c w_c \mathbf{G}_c^\top \mathbf{G}_c)}_{\mathbf{A}} \mathbf{x} = \frac{1}{h^2}\mathbf{M}\mathbf{y} + \sum_c w_c \mathbf{G}_c^\top \mathbf{p}_c. \tag{4.15}$$

It is easy to see that each local and global step ensures $\tilde{g}$ is non-increasing. Since $\tilde{g}$ is bounded below by 0, PD guarantees to converge to a local minimum of $\tilde{g}$ satisfying the gradient condition $\nabla_\mathbf{x}\tilde{g} = \mathbf{0}$. Interestingly, Liu et al. [96] establishes that $\nabla_\mathbf{x}\tilde{g} = \nabla g$ upon convergence, confirming that the solution from PD is indeed a critical point of g that solves the implicit time integration. We summarize the local-global solver in Alg. 2, which serves as a basis for our contact handling algorithm to be described in Sec. 4.1.4.

The source of efficiency in *forward* PD simulation lies in the fact that $\mathbf{A}$ in the global step is a constant, symmetric positive definite matrix. Therefore, the Cholesky factorization of $\mathbf{A}$ can be precomputed, after which each global step requires back-substitution only. In the next section, we will show that we can also use $\mathbf{A}$ in *back-propagation* to obtain significant speedup.

4.1.3 Backpropagation

We now describe our PD-based backpropagation method. Our key observation is that the bottleneck in backpropagation lies in the computation of $\nabla^2 g(\mathbf{x})$ in Eqn. (4.10). Following the same idea as in forward PD simulation, we propose to decouple $\nabla^2 g(\mathbf{x})$ into a global, constant matrix and a local, massively parallelizable nonlinear component. To see this point, we compute $\nabla^2 E$ using Eqns. (4.12) and (4.13):

$$\nabla E(\mathbf{x}) = \sum_c w_c \mathbf{G}_c^\top (\mathbf{G}_c \mathbf{x} - \mathbf{p}_c), \tag{4.16}$$

$$\nabla^2 E(\mathbf{x}) = \sum_c w_c \mathbf{G}_c^\top \mathbf{G}_c - \sum_c w_c \mathbf{G}_c^\top \frac{\partial \mathbf{p}_c}{\partial \mathbf{x}}. \tag{4.17}$$

Note that in Eqn. (4.16), $\frac{\partial \mathbf{p}_c}{\partial \mathbf{x}}$ can be safely ignored according to the envelope theorem (see Appendix in [96]). According to Eqn. (4.11), $\nabla^2 g(\mathbf{x})$ now becomes:

$$\nabla^2 g(\mathbf{x}) = \frac{1}{h^2}\mathbf{M} + \sum_c w_c \mathbf{G}_c^\top \mathbf{G}_c - \underbrace{\sum_c w_c \mathbf{G}_c^\top \frac{\partial \mathbf{p}_c}{\partial \mathbf{x}}}_{\Delta \mathbf{A}} = \mathbf{A} - \Delta\mathbf{A}. \tag{4.18}$$

It is now clear that $\Delta\mathbf{A}$ is the source of nonlinearity in $\nabla^2 g$. The matrix splitting of $\nabla^2 g = \mathbf{A} - \Delta\mathbf{A}$ suggests the following iterative solver for Eqn. (4.10):

$$\mathbf{A}\mathbf{z}^{k+1} = \Delta\mathbf{A}\mathbf{z}^k + (\frac{\partial L}{\partial \mathbf{x}})^\top, \tag{4.19}$$

where k indicates the iteration number. Therefore, we propose a local-global solver for Eqn. (4.10): At the k-th iteration, the local step computes $\Delta\mathbf{A}\mathbf{z}^k$ across all energies E_c, forming the right-hand vector in Eqn. (4.19). In the global step, we solve $\mathbf{z}^{k+1}$ by back-substituting $\mathbf{A}$. Note that $\mathbf{A}$ is the same constant matrix in forward PD simulation, so we can reuse the Cholesky factorization of $\mathbf{A}$. The source of efficiency in this local-global solver is similar to what PD proposes to speed up forward simulation: Essentially, this local-global solver trades the expensive matrix assembly and factorization of $\nabla^2 g$ in Eqn. (4.10) with iterations on a constant, prefactorized linear system of equations. We summarize our PD backpropagation algorithm in Alg. 3.

Convergence rate For any iterative algorithm design, the immediate follow-up questions are whether such an algorithm is guaranteed to converge, and, if so, how fast the convergence rate is. To answer these questions, we use $(\frac{\partial L}{\partial \mathbf{x}})^\top = \mathbf{A}\mathbf{z} - \Delta\mathbf{A}\mathbf{z}$ and Eqn. (4.19) to obtain

$$\mathbf{A}(\mathbf{z}^{k+1} - \mathbf{z}) = \Delta\mathbf{A}(\mathbf{z}^k - \mathbf{z}), \tag{4.20}$$

Algorithm 3: PD backpropagation in one time step

```
Input: $\mathbf{y}$, $\mathbf{x}$ (already computed in forward simulation), and $\frac{\partial L}{\partial \mathbf{x}}$;
Output: $\frac{\partial L}{\partial \mathbf{y}}$;
Initialize $\mathbf{z} = \mathbf{0}$;
while $\mathbf{z}$ not converged do
    $\mathbf{b} = \Delta\mathbf{A}\mathbf{z} + (\frac{\partial L}{\partial \mathbf{x}})^\top$;  // Local step parallelizing $\Delta\mathbf{A}\mathbf{z}$;
    $\mathbf{z} = \mathbf{A}^{-1}\mathbf{b}$;  // Global step;
$\frac{\partial L}{\partial \mathbf{y}} = \frac{1}{h^2}\mathbf{z}^\top\mathbf{M}$;  // Eqn. (4.9);
```

from which we conclude the error at the k-th iteration is $\|\mathbf{z}^k - \mathbf{z}\|_2 = \|(\mathbf{A}^{-1}\Delta\mathbf{A})^k(\mathbf{z}^0 - \mathbf{z})\|_2$. It follows that the iteration in Eqn. (4.19) is guaranteed to converge from any initial guess $\mathbf{z}^0$ if and only if $\rho(\mathbf{A}^{-1}\Delta\mathbf{A}) < 1$, where $\rho(\cdot)$ indicates the spectral radius of a matrix. It is challenging to provide more theoretical results on $\rho(\mathbf{A}^{-1}\Delta\mathbf{A})$ because it depends heavily on the specific form of E_c, which we leave as future work. In practice, we do not observe convergence issues with Eqn. (4.19) in any of our experiments, which seems to imply $\rho(\mathbf{A}^{-1}\Delta\mathbf{A}) < 1$ is likely to be satisfied.

Further acceleration with Quasi-Newton methods Inspired by Liu et al. [96] which apply the quasi-Newton method to speed up forward PD simulation, we now show that a similar numerical optimization perspective can also be applied to speed up our proposed local-global solver in backpropagation. Solving Eqn. (4.10) equals finding the critical point of the following energy $h(\mathbf{z})$:

$$h(\mathbf{z}) = \frac{1}{2}\mathbf{z}^\top\nabla^2 g(\mathbf{x})\mathbf{z} - \frac{\partial L}{\partial \mathbf{x}}\mathbf{z}. \tag{4.21}$$

It is easy to verify that $\nabla h(\mathbf{z}) = \mathbf{0}$ is essentially Eqn. (4.10). We stress that in backpropagation both $\nabla^2 g(\mathbf{x})$ and $\frac{\partial L}{\partial \mathbf{x}}$ are known values computed at $\mathbf{x}$ solved from forward simulation. If we apply Newton's method to this critical-point problem, the update rule will be as follows (see Eqns. (4.6) and (4.7) in Sec. 4.1.2):

$$\mathbf{z}^{k+1} = \mathbf{z}^k - [\nabla^2 h(\mathbf{z}^k)]^{-1}\nabla h(\mathbf{z}^k). \tag{4.22}$$

The true Hessian of h is $\nabla^2 g(\mathbf{x}) = \mathbf{A} - \Delta\mathbf{A}$ from Eqn. (4.18). If we approximate it with $\mathbf{A}$, we get the following quasi-Newton update rule:

$$\begin{aligned}\mathbf{z}^{k+1} &= \mathbf{z}^k - \mathbf{A}^{-1}\nabla h(\mathbf{z}^k)\\ &= \mathbf{z}^k - \mathbf{A}^{-1}[(\mathbf{A} - \Delta\mathbf{A})\mathbf{z}^k - (\frac{\partial L}{\partial \mathbf{x}})^\top]\\ &= \mathbf{A}^{-1}\Delta\mathbf{A}\mathbf{z}^k + \mathbf{A}^{-1}(\frac{\partial L}{\partial \mathbf{x}})^\top,\end{aligned} \tag{4.23}$$

which is identical to the iteration in Eqn. (4.19). As a result, the local-global solver we propose can be reinterpreted as running a simplified quasi-Newton method with a constant Hessian approximation $\mathbf{A}$. By applying a full quasi-Newton method, e.g., BFGS, we can reuse the Cholesky decomposition of $\mathbf{A}$ with little extra overhead of vector products and achieve a superlinear convergence rate [110]. Moreover, similar to the previous work [96], we can apply line search techniques to ensure convergence when $\rho(\mathbf{A}^{-1}\Delta\mathbf{A}) \geq 1$, even though we do not experience convergence issues in practice.

4.1.4 Contact Handling

We have described the basic framework of DiffPD in Sec. 4.1.3. In this section, we propose a novel method to incorporate contact handling and contact gradients into DiffPD. The challenges in developing such a contact handling algorithm are twofold: First, it must be compatible with our basic PD framework in *both* forward simulation *and* backpropagation. Second, it must support differentiability. In this section, we discuss two contact options that DiffPD supports: a penalty-based contact model with static and dynamic friction and a complementarity-based contact model supporting non-penetration conditions and static friction. Both options have their advantages and disadvantages: the penalty-based method is more straightforward to implement and easier to be integrated into a machine learning framework, e.g., as an explicit neural network layer in PyTorch. However, we find it typically requires a careful, scene-by-scene tuning of its parameters. On the other hand, our complementarity-based method does not rely on scene-dependent parameters, but it is currently limited to static friction only. Our penalty-based method is more suitable for tasks that favor speed and simplicity over physical accuracy, and the complementarity-based method is more useful when non-penetration conditions need to be strictly enforced and sliding motions are rare, e.g., simulating a wheeled robot.

Penalty-Based Contact

Previous papers on PD simulation typically handle contact forces with a penalty-based soft contact model [40, 39, 153, 154, 22, 96]. One common way to model contact forces is to add an additional, fictitious energy E_c with $\mathcal{M}_c$ being the contact surface and its exterior and $\mathbf{G}_c$ being a matrix so that $\mathbf{G}_c\mathbf{x}$ selects contact nodes from $\mathbf{x}$. This way, whenever a node penetrates the contact surface, E_c exerts a contact force that attempts to push it back to the contact surface. As such a contact model can be seamlessly integrated into PD forward simulation, our backpropagation method in Sec. 4.1.3 naturally supports it.

Handling static and dynamic frictional forces with a penalty-based model in PD is slightly trickier. Since friction is typically related to nodal velocities instead of nodal positions $\mathbf{x}$, it is not straightforward to find an E_c that characterizes it. Therefore, instead of modeling friction with an additional E_c, we take the penalty-based frictional forces described in Macklin et al. [100] and add them directly to $\mathbf{f}_{\text{ext}}$. Deriving gradients with respect to such frictional forces is still straightforward as we can easily compute $\frac{\partial L}{\partial \mathbf{f}_{\text{ext}}}$ using the chain rule and the relation $\frac{\partial \mathbf{y}}{\partial \mathbf{f}_{\text{ext}}} = h^2\mathbf{M}^{-1}$ after computing $\frac{\partial L}{\partial \mathbf{y}}$.

Complementarity-Based Contact

An alternative to penalty-based contact is to model contact and friction using complementarity constraints [100, 97]. Complementarity-based contact models are suitable for applications requiring high physical fidelity but typically require extra computational cost. Ly et al. [97] present a general framework for handling complementarity-based contact and friction in PD forward simulation. More concretely, Ly et al. [97] model contact and friction with the Signorini-Coulomb law and focus on applications in cloth simulation. Our approach is relevant to Ly et al. [97] but has substantial difference because our focus in this paper is on 3D volumetric deformable bodies, which typically have a sparse contact set, i.e., the nodes in contact during simulation is usually a small portion of the full set of nodes. Below we will present a differentiable, complementarity-based contact model that leverages such sparsity to gain speedup in both forward simulation and backpropagation. Our contact model ensures non-penetration conditions and, in exchange for speedup, handles static friction only. We leave differentiable, complementarity-based dynamic friction model as future work.

Contact model Let $\phi(\cdot) : \mathcal{R}^3 \to \mathcal{R}$ be the signed-distance function of the contact surface with $\phi < 0$ indicating the space occupied by the obstacle. We require the solution $\mathbf{x}$ to the implicit time integration to satisfy the following complementarity condition for any node indexed by j:

$$\begin{cases} \phi(\mathbf{x}_j) > 0, \mathbf{r}_j = \mathbf{0}, & (4.24a) \\ \text{or } \phi(\mathbf{x}_j) = 0, \mathbf{r}_{j|N} \geq 0, & (4.24b) \end{cases}$$

where $\mathbf{x}_j$ and $\mathbf{r}_j$ are 3D vectors indicating the nodal position and contact force of node j. The notation $\mathbf{r}_{j|N} \in \mathcal{R}$ is the normal component of $\mathbf{r}_j$ where the normal is computed from the contact surface ϕ at the contact location $\mathbf{x}_j$. In other words, for each node j, it must be either above the contact surface ($\phi(\mathbf{x}_j) > 0$) with zero contact force ($\mathbf{r}_j = \mathbf{0}$) or in contact ($\phi(\mathbf{x}_j) = 0$) with a positive contact force along the normal direction ($\mathbf{r}_{j|N} = 0$). The implicit time integration in Eqn. (4.4) now becomes:

$$\begin{cases} \frac{1}{h^2}\mathbf{M}(\mathbf{x} - \mathbf{y}) - \mathbf{f}_{\text{int}}(\mathbf{x}) = \mathbf{r}, & (4.25a) \\ (\mathbf{x}, \mathbf{r}) \text{ satisfy Eqn. (4.24)}, & (4.25b) \end{cases}$$

where the notation $\mathbf{r}$ stacks up all contact force $\mathbf{r}_j$ from each node j.

Remarks on friction Eqn. (4.25) does not fully constrain the solution $\mathbf{x}$ because, for any $\mathbf{x}_j$ in contact with $\phi = 0$, $\mathbf{x}_j$ can slide on $\phi = 0$ and $\mathbf{r}_j$ will compensate any force needed. This can be resolved by imposing additional location constraints on $\mathbf{x}_j$. Some common strategies include 1) in the penalty-based model before, $\mathbf{x}_j$ is chosen as certain projection onto $\phi = 0$, 2) gluing $\mathbf{x}_j$ to its original position at the beginning of the time step, and 3) setting it to the contact point computed from collision detection [28], which is usually the intersection between $\phi = 0$ and the ray

$\mathbf{x}_j + t\mathbf{v}_j, 0 \leq t \leq h$. Any of these strategies are compatible with DiffPD as long as they can compute a target location $\mathbf{x}_j^*$ on $\phi = 0$ if a collision detection algorithm indicates $\mathbf{x}_j$ is in contact with $\phi = 0$. In DiffPD, we choose the third strategy mentioned above, which essentially models a very sticky contact surface that provides infinitely large static friction once $\mathbf{x}_j$ is in contact.

Time integration with contact We first introduce some notations to better explain our solver to Eqn. (4.25). Let $\mathcal{I} = \{0, 1, 2, \cdots, n-1\}$ be the indices of all n nodes in the system. We use $\overline{\mathbf{S}}$ to denote the complement of a set $\mathbf{S} \subseteq \mathcal{I}$. In other words, $\mathbf{S}$ and $\overline{\mathbf{S}}$ is a two-set partition of $\mathcal{I}$. For any subsets $\mathbf{S}_r, \mathbf{S}_c \subseteq \mathcal{I}$, we use $\mathbf{A}_{\mathbf{S}_r \times \mathbf{S}_c}$ to indicate the submatrix of $\mathbf{A}$ created by keeping entries whose row and column indices are from nodes in $\mathbf{S}_r$ and $\mathbf{S}_c$, respectively. Similarly, for any vector $\mathbf{a}$, we define $\mathbf{a}_\mathbf{S}$ as the vector generated by keeping elements whose indices are from nodes in $\mathbf{S}$. For any vectors $\mathbf{a}$ and $\mathbf{b} \in \mathcal{R}^{3|\mathbf{S}|}$, We use $\mathbf{a}_{\mathbf{S}=\mathbf{b}}$ to indicate that $\mathbf{a}$ satisfies $\mathbf{a}_\mathbf{S} = \mathbf{b}$.

The high-level idea of our time integrator with the aforementioned contact model is described in Alg. 4, which modifies Alg. 2 to find $\mathbf{x}$ that satisfies Eqn. (4.25). We start with any collision detection algorithm that can propose a set of candidate contact nodes $\mathcal{C}$ and compute a target location $\mathbf{x}_j^*$ for any $j \in \mathcal{C}$. Next, we use the proposed $\mathcal{C}$ to split the complementarity condition in Eqn. (4.25b) and solve Eqn. (4.25a): For any $j \in \mathcal{C}$, we set $\mathbf{x}_j = \mathbf{x}_j^*$; for any $j \notin \mathcal{C}$, we set $\mathbf{r}_j = \mathbf{0}$. This makes Eqn. (4.25a) a balanced system with an equal number of equations and variables. Finally, we use the solved $\mathbf{x}$ to computed $\mathbf{r}_j$ at each $j \in \mathcal{C}$ and check if $\mathbf{r}_{j|N} \geq 0$ is satisfied. If $\mathbf{x}$ results in some negative $\mathbf{r}_{j|N}$, these nodes are removed from $\mathcal{C}$ and a new iteration begins with the updated $\mathcal{C}$. Similarly, if $\phi(\mathbf{x}_j)$ becomes negative, such a node j is added to $\mathcal{C}$. Essentially, we are running the active-set algorithm on Eqn. (4.25a) with linear constraints, and more advanced active set schemes can potentially be used to rebuild $\mathcal{C}$ more efficiently. Using the notations above, our algorithm attempts to solve the following reduced system at each iteration:

$$\frac{1}{h^2}\mathbf{M}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}(\mathbf{x}-\mathbf{y})_{\overline{\mathcal{C}}} - \mathbf{f}_{\text{int}}(\mathbf{x}_{\mathcal{C}=\mathbf{x}^*})_{\overline{\mathcal{C}}} = \mathbf{0}, \tag{4.26}$$

where $\mathbf{x}^*$ stacks up $\mathbf{x}_j^*$ for all $j \in \mathcal{C}$. Accordingly, the definition of g is updated as follows, which we rename as $g_\mathcal{C}$:

$$g_\mathcal{C}(\mathbf{x}_{\mathcal{C}=\mathbf{x}^*}) = \frac{1}{2h^2}(\mathbf{x}-\mathbf{y})_{\overline{\mathcal{C}}}^\top \mathbf{M}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}(\mathbf{x}-\mathbf{y})_{\overline{\mathcal{C}}} + E(\mathbf{x}_{\mathcal{C}=\mathbf{x}^*}). \tag{4.27}$$

It is easy to check that the left-hand side of Eqn. (4.26) is identical to $\nabla_{\mathbf{x}_{\overline{\mathcal{C}}}} g_\mathcal{C}$. Therefore, solving Eqn. (4.26) is equal to finding the critical point of this modified g function, and we can still apply Newton's method but with a slightly different definition of $\nabla^2 g$:

$$\nabla^2_{\mathbf{x}_{\overline{\mathcal{C}}}} g_\mathcal{C} = \frac{1}{h^2}\mathbf{M}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}} + (\nabla^2 E)_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}} = (\nabla^2 g)_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}. \tag{4.28}$$

In other words, $\nabla^2_{\mathbf{x}_{\overline{\mathcal{C}}}} g_\mathcal{C}$ is a submatrix of $\nabla^2 g$ in Eqn. (4.11) created by deleting rows

Algorithm 4: PD forward simulation with contact

```
Input: $\mathbf{y}$;
Output: $\mathbf{x}$ that satisfies Eqn. (4.25);
Run a collision detection algorithm to get $\mathcal{C}$ and $\mathbf{x}^*$;
while $\mathcal{C}$ not converged do
    Initialize $\mathbf{x} = \mathbf{y}$ and set $\mathbf{x}_\mathcal{C} = \mathbf{x}^*$;
    while $\mathbf{x}$ not converged do
        $\mathbf{p}_c = \arg\min_{\mathbf{p}_c \in \mathcal{M}_c} \tilde{E}_c(\mathbf{x}, \mathbf{p}_c)$; // Local step;
        $\mathbf{b} = \frac{1}{h^2}\mathbf{M}\mathbf{y} + \sum_c w_c \mathbf{G}_c^\top(\mathbf{p}_c - \mathbf{G}_c \mathbf{x}_{\mathcal{C}=\mathbf{x}^*, \overline{\mathcal{C}}=\mathbf{0}})$;
        Run Alg. 5 to solve $\mathbf{x}_{\overline{\mathcal{C}}} = (\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}})^{-1}\mathbf{b}_{\overline{\mathcal{C}}}$; // Global step;
    $\mathbf{r} = \frac{1}{h^2}\mathbf{M}(\mathbf{x} - \mathbf{y}) - \mathbf{f}_{\text{int}}(\mathbf{x})$; // Eqn. (4.25a);
    Update $\mathcal{C}$ based on $\mathbf{r}_j$, $\phi(\mathbf{x}_j)$, and Eqn. (4.24);
```

and columns from $\mathcal{C}$.

Implications on forward PD In the PD framework, $g_\mathcal{C}$ also induces a modified surrogate function $\tilde{g}$, which we rename as $\tilde{g}_\mathcal{C}$:

$$\tilde{g}_\mathcal{C}(\mathbf{x}_{\mathcal{C}=\mathbf{x}^*}, \mathbf{p}) = \frac{1}{2h^2}(\mathbf{x} - \mathbf{y})_{\overline{\mathcal{C}}}^\top \mathbf{M}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}(\mathbf{x} - \mathbf{y})_{\overline{\mathcal{C}}} + \sum_c \tilde{E}_c(\mathbf{x}_{\mathcal{C}=\mathbf{x}^*}, \mathbf{p}_c). \tag{4.29}$$

It is still true that the original local-global solver will ensure $\tilde{g}_\mathcal{C}$ is non-increasing and converge to a critical point of $g_\mathcal{C}$. With the constraint $\mathbf{x}_\mathcal{C} = \mathbf{x}^*$, the local step can project each $\mathbf{G}_c\mathbf{x}$ to obtain $\mathbf{p}_c$ as before. The global step, on the other hand, requires some modification, as can be best seen after computing $\nabla_{\mathbf{x}_{\overline{\mathcal{C}}}}\tilde{g}_\mathcal{C}$:

$$\nabla_{\mathbf{x}_{\overline{\mathcal{C}}}}\tilde{g}_\mathcal{C} = \frac{1}{h^2}\mathbf{M}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}(\mathbf{x} - \mathbf{y})_{\overline{\mathcal{C}}} + \sum_c w_c(\mathbf{G}_c^\top)_{\overline{\mathcal{C}}\times\mathcal{I}}(\mathbf{G}_c\mathbf{x} - \mathbf{p}_c). \tag{4.30}$$

Setting $\nabla_{\mathbf{x}_{\overline{\mathcal{C}}}}\tilde{g}_{\overline{\mathcal{C}}} = \mathbf{0}$ and using the fact that $\mathbf{x}_\mathcal{C} = \mathbf{x}^*$, we obtain the new global step with a linear system modified from Eqn. (4.15):

$$\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}\mathbf{x}_{\overline{\mathcal{C}}} = [\frac{1}{h^2}\mathbf{M}\mathbf{y} + \sum_c w_c \mathbf{G}_c^\top(\mathbf{p}_c - \mathbf{G}_c\mathbf{x}_{\mathcal{C}=\mathbf{x}^*, \overline{\mathcal{C}}=\mathbf{0}})]_{\overline{\mathcal{C}}}, \tag{4.31}$$

where $\mathbf{x}_{\mathcal{C}=\mathbf{x}^*, \overline{\mathcal{C}}=\mathbf{0}}$ is a vector satisfying $\mathbf{x}_\mathcal{C} = \mathbf{x}^*$ and $\mathbf{x}_{\overline{\mathcal{C}}} = \mathbf{0}$. Although the right-hand side seems complicated, it can still be parallelized across all E_c. It is the left-hand side matrix $\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}$ that deserves more attention: Since $\mathcal{C}$ is a set that changes dynamically between each time step, $\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}$ randomly erases different rows and columns from $\mathbf{A}$, which means the Cholesky factorization of $\mathbf{A}$ no longer applies. Our key observation is that $\mathcal{C}$ is usually a small subset of full nodes in 3D volumetric deformable bodies. This allows us to formulate row and column deletions on $\mathbf{A}$ as a *low-rank update*, from which we derive efficient solvers that can reuse the Cholesky factorization of $\mathbf{A}$.

Low-rank update Define a permutation σ on $\mathcal{I}$ with the following property: σ shuffles $\mathcal{I}$ so that indices from $\mathcal{C}$ come before those in $\overline{\mathcal{C}}$ and the internal orders inside $\mathcal{C}$ and $\overline{\mathcal{C}}$ are preserved. Define $\mathbf{P}$ as the corresponding permutation matrix: $\mathbf{P}_{ij} = 1$ if $\sigma(i) = j$ and 0 otherwise. Now $\mathbf{AP}$ shuffles all columns of $\mathbf{A}$ so that the i-th column in $\mathbf{A}$ becomes the $\sigma(i)$-th column in $\mathbf{AP}$. Similarly, $\mathbf{P}^\top\mathbf{A}$ shuffles all rows of $\mathbf{A}$ in the same way. We now rewrite $\mathbf{P}^\top\mathbf{AP}$ as a 2×2 block matrix:

$$\mathbf{P}^\top\mathbf{AP} = \begin{pmatrix} \mathbf{A}_{\mathcal{C}\times\mathcal{C}} & \mathbf{A}_{\mathcal{C}\times\overline{\mathcal{C}}} \\ \mathbf{A}_{\overline{\mathcal{C}}\times\mathcal{C}} & \mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}} \end{pmatrix}. \tag{4.32}$$

Let $c = |\mathcal{C}|$ and define $\mathbf{U} \in \mathcal{R}^{3n\times 2c}$ as follows:

$$\mathbf{U} = \begin{pmatrix} \mathbf{U}_L & \mathbf{U}_R \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{\overline{\mathcal{C}}\times\mathcal{C}} \end{pmatrix}, \tag{4.33}$$

where $\mathbf{U}_L, \mathbf{U}_R \in \mathcal{R}^{3n\times c}$ represent the left and right half of $\mathbf{U}$ and $\mathbf{I}$ is the identity matrix of a proper size. Similarly, we define $\mathbf{V} \in \mathcal{R}^{2c\times 3n}$ as follows:

$$\mathbf{V} = \begin{pmatrix} \mathbf{U}_R^\top \\ \mathbf{U}_L^\top \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{A}_{\mathcal{C}\times\overline{\mathcal{C}}} \\ \mathbf{I} & \mathbf{0} \end{pmatrix}. \tag{4.34}$$

It is now easy to verify that the product of $\mathbf{UV}$ is the following low-rank matrix:

$$\mathbf{UV} = \begin{pmatrix} \mathbf{0} & \mathbf{A}_{\mathcal{C}\times\overline{\mathcal{C}}} \\ \mathbf{A}_{\overline{\mathcal{C}}\times\mathcal{C}} & \mathbf{0} \end{pmatrix}, \tag{4.35}$$

and subtracting it from $\mathbf{P}^\top\mathbf{AP}$ results in a block-diagonal matrix:

$$\mathbf{P}^\top\mathbf{AP} - \mathbf{UV} = \mathbf{P}^\top \underbrace{(\mathbf{A} - \mathbf{PUVP}^\top)}_{\mathbf{A_P}} \mathbf{P} = \begin{pmatrix} \mathbf{A}_{\mathcal{C}\times\mathcal{C}} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}} \end{pmatrix}. \tag{4.36}$$

Therefore, we can obtain $(\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}})^{-1}$ by inverting $\mathbf{P}^\top\mathbf{AP} - \mathbf{UV}$. Since inverting $\mathbf{P}$ is trivial ($\mathbf{P}^{-1} = \mathbf{P}^\top$), we focus on explaining how to obtain $\mathbf{A}_\mathbf{P}^{-1}$:

$$\mathbf{A}_\mathbf{P}^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{PU}(\mathbf{I} - \mathbf{VP}^\top\mathbf{A}^{-1}\mathbf{PU})^{-1}\mathbf{VP}^\top\mathbf{A}^{-1}. \tag{4.37}$$

Since $\mathbf{A}$ is prefactorized, operations using $\mathbf{A}^{-1}$ in the matrix identity above can be executed efficiently. Moreover, with the assumption that $c \ll n$, $\mathbf{I} - \mathbf{VP}^\top\mathbf{A}^{-1}\mathbf{PU} \in \mathcal{R}^{2c\times 2c}$ is a small matrix compared to $\mathbf{A}$, and inverting it (solving a linear system whose left-hand side is this matrix) can be done efficiently. Putting everything together, we transform the problem of factorizing $\mathbf{A_P}$ into factorizing a much smaller linear system of equations.

Time complexity We now consider a brute-force implementation of Eqn. (4.37) and analyze its time complexity. The time cost is dominated by computing $\mathbf{A}^{-1}\mathbf{PU}$

which takes $\mathcal{O}(n^2c)$ time. While this is still asymptotically smaller than the cost of factorizing the modified matrix, which generally takes $\mathcal{O}(n^3)$ time, the speedup in practice may not be as much as predicted by the time analysis due to the sparsity of $\mathbf{A_P}$. Therefore, further simplification on Eqn. (4.37) would still be desirable.

Further acceleration To reduce the time cost of computing $\mathbf{A}^{-1}\mathbf{PU}$, we notice that $\mathbf{PU}$ shuffles all rows of $\mathbf{U}$ with the inverse mapping σ^{-1}. As a result, $\mathbf{PU}_L$, the left part of $\mathbf{PU}$, is effectively $\mathbf{I}_{\mathcal{I}\times\mathcal{C}}$, i.e., a collection of one-hot column vectors $\mathbf{e}_j, j \in \mathcal{C}$, where the j-th entry in $\mathbf{e}_j$ is 1. This means that we can precompute $\mathbf{A}^{-1}\mathbf{I}_{\mathcal{I}\times\mathcal{C}}$ using a maximum possible $\mathcal{C}$ (e.g., all surface nodes) before the whole simulation begins and look up $\mathbf{A}^{-1}\mathbf{e}_j, j \in \mathcal{C}$ on the fly.

It turns out that the same idea can also be used for computing $\mathbf{A}^{-1}\mathbf{PU}_R$, the right half of the solution, with a slight modification. Notice that $\mathbf{PU}_R$ can be obtained from $\mathbf{A}$ by fetching $\mathbf{A}_{\mathcal{I}\times\mathcal{C}}$ and zeroing out corresponding rows in $\mathcal{C}$:

$$\mathbf{PU}_R = \mathbf{A}_{\mathcal{I}\times\mathcal{C}} - \mathbf{I}_{\mathcal{I}\times\mathcal{C}}\mathbf{A}_{\mathcal{C}\times\mathcal{C}}. \tag{4.38}$$

We can, therefore, compute $\mathbf{A}^{-1}\mathbf{PU}_R$ as follows:

$$\mathbf{A}^{-1}\mathbf{PU}_R = \mathbf{A}^{-1}\mathbf{A}_{\mathcal{I}\times\mathcal{C}} - \mathbf{A}^{-1}\mathbf{I}_{\mathcal{I}\times\mathcal{C}}\mathbf{A}_{\mathcal{C}\times\mathcal{C}} = \mathbf{I}_{\mathcal{I}\times\mathcal{C}} - \mathbf{A}^{-1}\mathbf{I}_{\mathcal{I}\times\mathcal{C}}\mathbf{A}_{\mathcal{C}\times\mathcal{C}}. \tag{4.39}$$

Since $\mathbf{A}^{-1}\mathbf{I}_{\mathcal{I}\times\mathcal{C}}$ has been precomputed, the time complexity will be bounded by the matrix multiplication $\mathcal{O}(nc^2)$. Moreover, noting that $\mathbf{A}^{-1}$ is symmetric and $\mathbf{V}$ can be obtained from $\mathbf{U}$ by swapping and transposing block matrices $\mathbf{U}_L$ and $\mathbf{U}_R$, the results derived here can also be reused to assemble $\mathbf{VP}^\top\mathbf{A}^{-1}$.

In conclusion, we have reduced the time complexity of computing $\mathbf{A}^{-1}\mathbf{PU}$ from $\mathcal{O}(n^2c)$ to $\mathcal{O}(nc^2)$. Since the remaining operations, excluding solving $\mathbf{A}^{-1}$ in Eqn. (4.37), are also bounded by $\mathcal{O}(nc^2)$, we now have reduced the *overhead* of applying Eqn. (4.37) from $\mathcal{O}(n^2c)$ to $\mathcal{O}(nc^2)$, with the overhead defined as the extra cost brought by Eqn. (4.37) in addition to one linear solve $\mathbf{A}^{-1}$ with any right-hand side vector. We present the complete algorithm in pseudocode in Alg. 5, which serves as a subroutine in Alg. 4. We use $\mathbf{B}_k$ and $\mathbf{a}_k$ to denote intermediate matrices and vectors respectively, with the subscript k indicating the order of their first occurrence.

Backpropagation With a contact set $\mathcal{C}$ and the corresponding $\mathbf{x}^*$, the backpropagation scheme in Sec. 4.1.3 also needs modifications. Backpropagating from $\frac{\partial L}{\partial \mathbf{x}}$ to $\frac{\partial L}{\partial \mathbf{y}}$ now becomes trickier due to the existence of $\mathcal{C}$ and $\mathbf{x}^*$ from a collision detection algorithm, which splits both $\mathbf{x}$ and $\mathbf{y}$ into two vectors $\mathbf{x}_\mathcal{C}$, $\mathbf{x}_{\overline{\mathcal{C}}}$, $\mathbf{y}_\mathcal{C}$, and $\mathbf{y}_{\overline{\mathcal{C}}}$. Here, we will sketch the core idea by showing how gradients can be backpropagated from $\frac{\partial L}{\partial \mathbf{x}_{\overline{\mathcal{C}}}}$ to $\frac{\partial L}{\mathbf{y}_{\overline{\mathcal{C}}}}$. Backpropagation through other dependencies is easier to derive and therefore skipped.

From Eqns. (4.26) and (4.27), we see that $\mathbf{x}_{\overline{\mathcal{C}}}$ and $\mathbf{y}_{\overline{\mathcal{C}}}$ are constrained by $\nabla_{\mathbf{x}_{\overline{\mathcal{C}}}} g_\mathcal{C} = \mathbf{0}$.

Algorithm 5: Global step in Alg. 4.

```
Input: $\mathcal{C} \subseteq \mathcal{I}$, $\mathbf{x}^* \in \mathcal{R}^{3|\mathcal{C}|}$, and $\mathbf{b} \in \mathcal{R}^{3n}$;
Output: $\mathbf{x}$ such that $\mathbf{x}_{\mathcal{C}} = \mathbf{x}^*$ and $\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}\mathbf{x}_{\overline{\mathcal{C}}} = \mathbf{b}_{\overline{\mathcal{C}}}$;
Collect $\mathbf{B}_1 = \mathbf{A}^{-1}\mathbf{I}_{\mathcal{I}\times\mathcal{C}}$ from precomputed data;
$\mathbf{B}_2 = \mathbf{I}_{\mathcal{I}\times\mathcal{C}} - \mathbf{B}_1\mathbf{A}_{\mathcal{C}\times\mathcal{C}}$;
$\mathbf{B}_3 = (\mathbf{B}_1, \mathbf{B}_2)$;  // $\mathbf{B}_3 = \mathbf{A}^{-1}\mathbf{P}\mathbf{U}$;
// $\mathbf{V}\mathbf{P}^\top$ can be fetched from $\mathbf{A}$ without permutation;
// No need to compute $\mathbf{P}$;
$\mathbf{B}_4 = \mathbf{I} - \mathbf{V}\mathbf{P}^\top\mathbf{B}_3$;
Solve $\mathbf{a}_1$ from $\mathbf{A}\mathbf{a}_1 = \mathbf{b}$;
$\mathbf{a}_2 = (\mathbf{b}^\top\mathbf{B}_2, \mathbf{b}^\top\mathbf{B}_1)$;  // Row vector;
Solve $\mathbf{a}_3$ from $\mathbf{B}_4\mathbf{a}_3 = \mathbf{a}_2^\top$;
$\mathbf{x} = \mathbf{a}_1 + \mathbf{B}_3\mathbf{a}_3$;
Set $\mathbf{x}_{\mathcal{C}} = \mathbf{x}^*$;
```

By differentiating Eqn. (4.26) we obtain:

$$\nabla^2_{\mathbf{x}_{\overline{\mathcal{C}}}} g_{\mathcal{C}} \frac{\partial \mathbf{x}_{\overline{\mathcal{C}}}}{\partial \mathbf{y}_{\overline{\mathcal{C}}}} - \frac{1}{h^2}\mathbf{M}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}} = \mathbf{0}, \tag{4.40}$$

which is a reduced version of Eqn. (4.8). The chain rule still applies in a similar way:

$$\frac{\partial L}{\partial \mathbf{y}_{\overline{\mathcal{C}}}} = \frac{\partial L}{\partial \mathbf{x}_{\overline{\mathcal{C}}}}\frac{\partial \mathbf{x}_{\overline{\mathcal{C}}}}{\partial \mathbf{y}_{\overline{\mathcal{C}}}} = \frac{1}{h^2}\underbrace{\frac{\partial L}{\partial \mathbf{x}_{\overline{\mathcal{C}}}}[(\nabla^2 g)_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}]^{-1}}_{\mathbf{z}^\top}\mathbf{M}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}, \tag{4.41}$$

where a new adjoint vector $\mathbf{z}$ is defined. It should now become very clear that $\mathbf{z}$ is obtained from the following linear system of equations:

$$(\nabla^2 g)_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}\mathbf{z} = (\frac{\partial L}{\partial \mathbf{x}_{\overline{\mathcal{C}}}})^\top. \tag{4.42}$$

Now using Eqn. (4.18), we see the iterative solver in Sec. 4.1.3 becomes:

$$\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}\mathbf{z}^{k+1} = \Delta\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}\mathbf{z}^k + (\frac{\partial L}{\partial \mathbf{x}_{\overline{\mathcal{C}}}})^\top, \tag{4.43}$$

from which we see a similar issue we experience in forward simulation: $\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}$ changes dynamically, so the Cholesky factorization of $\mathbf{A}$ is not directly applicable. This is exactly where we can use the same global solver in Alg. 5 to retain the source of efficiency in our PD backpropagation algorithm. We summarize this new backpropagation method in Alg. 6.

Summary In summary, we have presented a differentiable contact handling algorithm that ensures non-penetration conditions and imposes infinitely large static

Algorithm 6: PD backpropagation with contact

```
Input: $\mathbf{y}$, $\mathbf{x}$ and $\mathcal{C}$ (from forward simulation), and $\frac{\partial L}{\partial \mathbf{x}_{\overline{\mathcal{C}}}}$;
Output: $\frac{\partial L}{\partial \mathbf{y}_{\overline{\mathcal{C}}}}$;
Initialize $\mathbf{z} = \mathbf{0}$;
while $\mathbf{z}$ not converged do
    $\mathbf{b} = (\Delta\mathbf{A})_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}\mathbf{z} + (\frac{\partial L}{\partial \mathbf{x}_{\overline{\mathcal{C}}}})^\top$;  // Local step;
    Run Alg. 5 to solve $\mathbf{z} = (\mathbf{A}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}})^{-1}\mathbf{b}$;  // Global step;
$\frac{\partial L}{\partial \mathbf{y}_{\overline{\mathcal{C}}}} = \frac{1}{h^2}\mathbf{z}^\top\mathbf{M}_{\overline{\mathcal{C}}\times\overline{\mathcal{C}}}$;  // Eqn. (4.9);
```

friction. Moreover, we have also discussed its implementation in forward simulation and backpropagation that can still benefit from the Cholesky factorization of **A**. We stress that there exist more physically accurate contact handling algorithms that satisfy not only non-penetration conditions but also the Coulomb's law of friction [28, 97, 90]. However, our contact handling algorithm achieves a good trade-off between differentiability, physically plausibility, and compatibility with our differentiable PD framework.

4.1.5 Evaluation

In this section, we compare DiffPD with a few baseline differentiable simulation methods and conduct ablation studies on the acceleration techniques in Sec. 4.1.3 and Sec. 4.1.4. We start by discussing the difference between implicit and explicit time-stepping schemes in backpropagation. Next, we compare our simulator with two other fully implicit simulators implemented with the Newton's method. We end this section with a discussion on the two contact models implemented in DiffPD. The end goal of this section is to evaluate the difference between different time-stepping methods and understand the source of efficiency in DiffPD. We implement both baseline algorithms and DiffPD in C++ and use Eigen [61] for sparse matrix factorization and linear solvers. We run all experiments in this section and next section on a virtual machine instance from Google Cloud Platform with 16 Intel Xeon Scalable Processors (Cascade Lake) @ 3.1 GHz and 64 GB memory. We use OpenMP for parallel

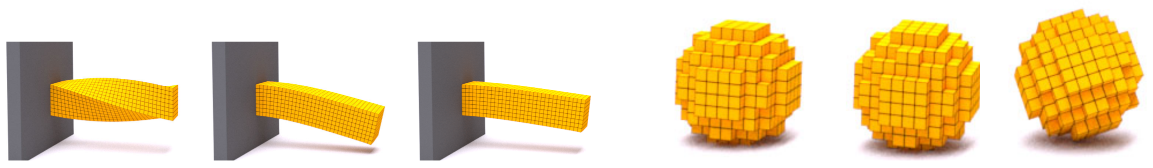


Figure 4-2: The "Cantilever" and "Rolling sphere" examples in Sec. 4.1.5 designed for comparing DiffPD to the Newton's method. The "Cantilever" example starts with a twisted cantilever (left), oscillates, and bends downwards eventually due to gravity. In the "Rolling sphere" example, we roll a soft sphere on the ground (right) which constantly breaks and reestablishes contact.

computing and 8 threads by default unless otherwise specified.

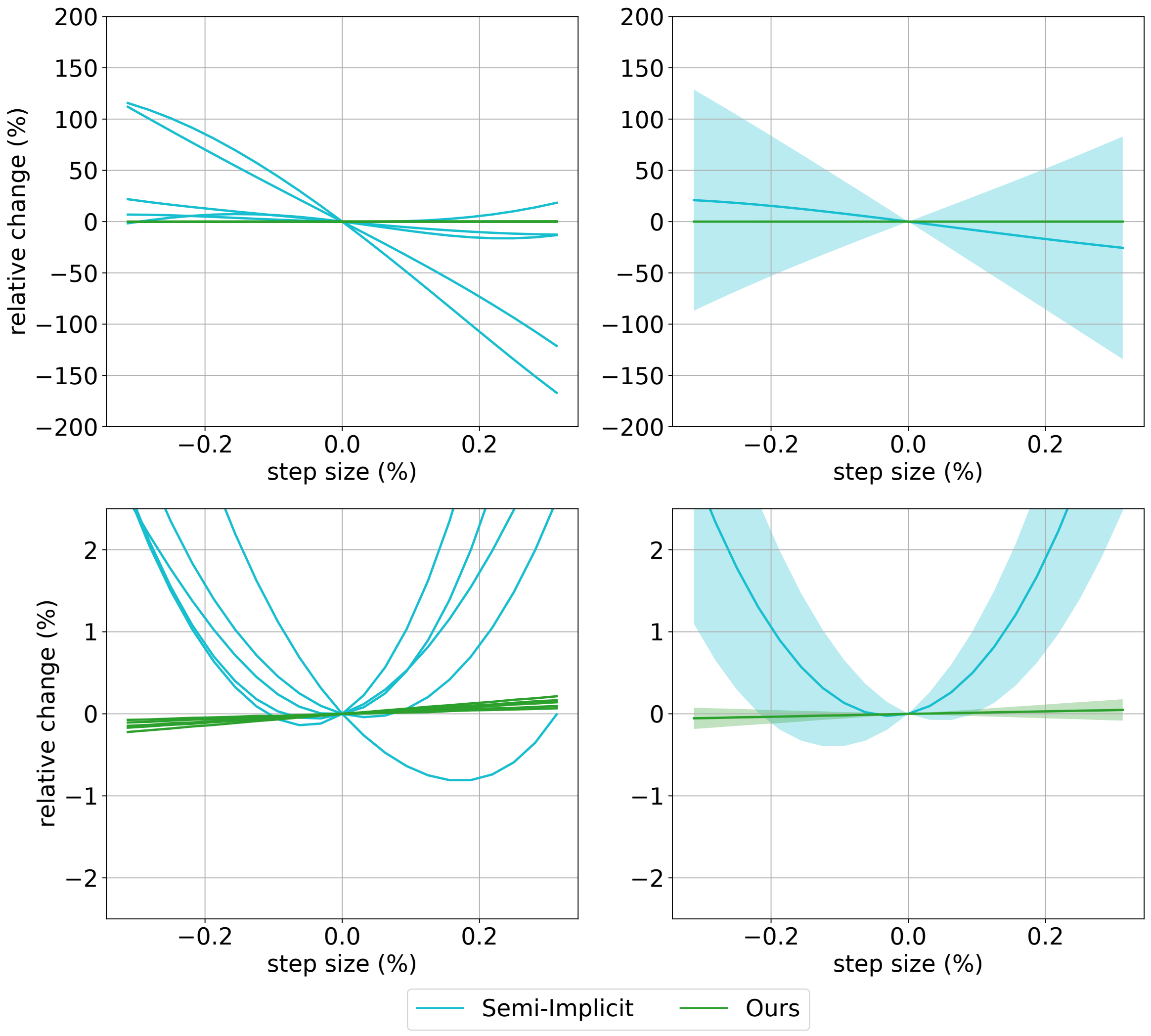


Figure 4-3: The relative changes in both the loss (top left) and the magnitude of the gradient (bottom left) for the explicit method (cyan) and our method (green) for 5 out of the 16 random directions in the neighborhood of the initial nodal positions $\mathbf{x}_0$. Also shown are the means (solid curves) and standard deviations (shaded) of the percent change in loss (top right) and the magnitude of the gradient (bottom right) for all 16 random directions.

Comparisons with Explicit Method

Compared with explicit time-stepping methods used in previous papers on differentiable simulation [69, 68, 136], implicit time integration brings two important changes to a differentiable simulator: First, implicit methods enable a much larger time step during simulation, resulting in much fewer number of frames. This is particularly beneficial for solving an inverse problem with a long time horizon as we store fewer states (nodal positions and velocities) in memory during backpropagation. Second, due to implicit damping, we can expect the landscape of the loss function defined on nodal states and their derived quantities to be smoother.

To demonstrate the memory consumption, we consider a soft cantilever discretized into $12 \times 3 \times 3$ hexahedral elements (a low-resolution version of the "Cantilever" example in Fig. 4-2 left). We impose Dirichlet boundary conditions on one end of the cantilever and simulate its vibration after twisting the other end of the cantilever under gravity for 0.2 seconds. We define a loss function L as a randomly generated weighted average of the final nodal positions and velocities. The implicit time integration in our simulator allows us to use a time step as large as 10 milliseconds, while a explicit implementation is only numerically stable in both forward simulation and backpropagation for time steps of 0.5 milliseconds. Since memory consumption during backpropagation is proportional to the number of frames, we can expect a $20\times$ increase in memory consumption for the explicit method, requiring additional techniques like checkpoint states [69, 136] before the problem size can be scaled up.

To demonstrate the influences of time-stepping schemes on the smoothness of the energy landscape, we visualize in Fig. 4-3 the loss function L and its gradient norm $|\nabla L|$ sliced along 16 random directions in the neighborhood of the cantilever's initial nodal positions. Specifically, let $\mathbf{x}_0$ be the initial nodal positions and let $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_{16}$ be the random directions. We plot $L(\mathbf{x}_0 + \alpha\mathbf{r}_i)$ and $|\nabla L(\mathbf{x}_0 + \alpha\mathbf{r}_i)|$ for each $\mathbf{r}_i$ (Fig. 4-3 left) with α being the step size, which is uniformly sampled between -0.3% and 0.3% of the cantilever beam length. The standard deviations from 16 random directions (Fig. 4-3 right) indicate that small perturbations in $\mathbf{x}_0$ lead to much smoother loss and gradients when implicit time integration is applied, which is not surprising due to numerical damping. From the perspective of differentiable simulation, a smoother energy landscape can be more favorable as it induces more well-defined gradients to be used by gradient-based optimization techniques.

Comparisons with Other Implicit Methods

We now compare our simulator with other implicit time-stepping schemes to evaluate its speedup in both forward and backward modes. We choose Newton's method with two standard sparse linear solvers: an iterative solver using preconditioned conjugate gradient (Newton-PCG) and a direct solver using Cholesky decomposition (Newton-Cholesky), as our baseline solvers for implicit time integration in Eqn. (4.3). First, we compare with the Newton's method in Sec. 4.1.3 without contact. Then, we benchmark the performance of the contact handling algorithm in Sec. 4.1.4. We reiterate that just like in the standard PD framework, any resultant speedup from DiffPD over Newton's method is under the assumption that the material model has a quadratic energy function. We extend our discussion to general hyperelastic materials at the end of paper and leave it as future work.

Simulation without Contact We benchmark our method, Newton-PCG, and Newton-Cholesky using a cantilever with $32 \times 8 \times 8$ elements, 8019 DoFs, and 243 Dirichlet boundary constraints ("Cantilever" in Fig. 4-2 and Table 4.2). The example runs for 25 frames with time steps of 10 milliseconds. We define the loss L as a randomly generated weighted sum of the final nodal positions and velocities.

In terms of the running-time comparison, we report results in Fig. 4-4 from running all three methods with 2, 4, and 8 threads and a range of convergence threshold (from 1e-1 to 1e-7) on the relative error in solving Eqn. (4.3). The speedup from parallel computing is less evident in the Newton's method because the majority of their computation time is spent on matrix refactorization – a process that cannot be trivially parallelized in Eigen. We conclude that our simulator has a clear advantage over Newton's method on the time cost of both forward simulation and backpropagation. For forward simulation, the speedup is well understood and discussed in many previous PD papers [22, 96]. For moderate tolerances (1e-3 to 1e-5), we observe a speedup of 9-16 times in forward simulation with 8 threads and note that it becomes less significant as precision increases. Both of these observations agree with previous work on PD for forward simulation. In backpropagation, DiffPD method is 6-13 times faster than Newton's method for moderate tolerances due to the reuse of the Cholesky decomposition and the quasi-Newton update. Specifically, we point out that without the proposed acceleration technique with quasi-Newton methods in Sec. 4.1.3, PD backpropagation is faster than Newton's method only for very low precision (orange in Fig. 4-4 right), confirming the necessity of the quasi-Newton updates.

Since PD is an iterative method whose result is dependent on the convergence threshold, it is necessary to justify which threshold is the most proper. To analyze the influence of the choice of thresholds, we use results from Newton-Cholesky as the oracle because it is a direct solver whose solution is computed with the machine precision in Eigen. We then compare both our method and Newton-PCG to the oracle by computing the loss and gradients of the "Cantilever" example with varying convergence thresholds and analyze when the results from the three methods start to coincide. This comparison provides quantitative guidance on the choice of convergence threshold and reveal the range in which our method can be a reliable alternative to the Newton's method in optimization tasks. We report our findings in Fig. 4-4. As Newton-PCG and our method are iterative methods, their accuracy improves when the convergence threshold becomes tighter. It can be seen from the figure that our method agrees with the Newton's method on the numerical losses and gradients when using a threshold as large as 1e-4. Therefore, we use 1e-4 as our default threshold in all applications to be discussed below unless otherwise specified. Referring back to Fig. 4-4, using 8 threads and with a convergence threshold of 1e-4, our method achieves significant speedup (12-16 times faster in forward simulation and 6.5-9 times faster in backpropagation) compared with Newton-PCG and Newton-Cholesky.

Simulation with Contact To create a benchmark scene that requires contact handling constantly, we roll a soft sphere on a horizontal collision plane for 100 frames with a time step of 5 milliseconds ("Rolling sphere" in Fig. 4-2 and Table 4.2). The sphere is voxelized into 552 elements with 2469 DoFs, and the maximum possible contact set $\mathcal{C}$ we consider consists of 72 nodes (216 DoFs) on the surface of the sphere. Similar to the "Cantilever" example, we define the loss function L as a randomly generated weighted average of the final nodal positions and velocities. We implement the contact handling algorithm in Sec. 4.1.4 with Newton-PCG, Newton-Cholesky, and our method, and we report their time cost as well as their loss and gradients in Fig. 4-5. It can be seen from Fig. 4-5 that the results from three methods start to converge when the convergence threshold reaches 1e-6, with which our method is 10 times faster than the Newton's method in both forward and backward mode (Fig. 4-5). Such speedup mainly comes from the low-rank update algorithm (Alg. 5) which avoids the expensive matrix factorization from scratch. Additionally, by comparing the orange and green curves in Fig. 4-5, we conclude that the acceleration technique of caching $\mathbf{A}^{-1}\mathbf{I}_{\mathcal{I}\times\mathcal{C}}$ further speeds up DiffPD by 25% in forward mode and 44% in backward mode when measured with 8 threads and a convergence threshold of 1e-6.

Ablation Study

We end this section with an ablation study on multiple components in our algorithm. We start with an empirical analysis on the iterative solver and the line search algorithm in our backpropagation algorithm (Sec. 4.1.3), followed by an evaluation on the penalty-based and the complementarity-based contact models.

Spectral radius and line search One key assumption we have made in our backpropagation solver is that the spectral radius of $\rho(\mathbf{A}^{-1}\Delta\mathbf{A}) < 1$, which is also one of the primary reasons why we have employed the line search algorithm as a safeguard when the assumption does not hold. Here, we use the "Cantilever" example check if this assumption holds empirically. We explicitly calculate $\rho(\mathbf{A}^{-1}\Delta\mathbf{A})$ we experience in "Cantilever" and observe a maximum value of 0.996, indicating that we can expect convergence in the iterative solver, which we further confirm by testing the iterative solver with 100 randomly generated, artificial right-hand side vector $\frac{\partial L}{\partial \mathbf{x}}$. We observe similar results about the convergence of the backpropagation solver in the "Rolling sphere" example as well as in our applications to be described in Sec. 4.1.6, indicating that it seems safe to expect the iterative solver to converge in practice despite the lack of a theoretical guarantee on it.

As employing line searches in our algorithm serves as a safeguard to cases when $\rho(\mathbf{A}^{-1}\Delta\mathbf{A}) > 1$, an implication from the observations on the spectral radius is that we rarely trigger line searches to reduce the step size in practice. In fact, in this "Cantilever" example, and in almost all applications below, we notice that the default step size (1 in Newton's and quasi-Newton methods) almost always allows us to skip the line search stage. Still, we precautionarily set the maximum number of line search iterations to be 10 for all examples.

Penalty-based contact We implement the penalty-based contact and frictional forces from Macklin et al. [100] in DiffPD and analyze them in both forward simulation and backpropagation. First, we use a standard "Slope" test with varying frictional coefficients in the penalty-based model to understand the expressiveness of this contact model in forward simulation. Second, we use a "Duck" example which optimizes frictional coefficients using the gradients of this contact model in backpropagation.

To show the capacity of the penalty-based contact model in forward simulation, we consider the "Slope" test visualized in Fig. 4-6. We place a squishy rubber duck (16776 DoFs and 24875 tetrahedrons) on four slopes with varying frictional coefficients from the penalty form in Macklin et al. [100] and let it slide for two seconds under gravity. We can see from the figure that with decreasing sliding friction from the left slope to the right slope, the implementation of Macklin et al. [100] in DiffPD generates different sliding distances that match our expectation qualitatively.

Backpropagating a penalty-based contact model is straightforward because it only requires a procedural application of chain rules to differentiate the penalty energy. To show the penalty-based model is fully compatible with DiffPD's backpropagation and can be useful in optimization problems, we design a "Duck" example (Fig. 4-7) with the same rubber duck but on a curved slide with frictional coefficients to be optimized (3 DoFs in total). The duck slides off the curved surface and aims to land on a target location (indicated by the white circle). The frictional coefficients affect the stickiness of the curve surface and control the exiting velocity of the duck when it leaves the slide, which further determines its movement under gravity afterwards. From the two motion sequences in Fig. 4-7 before and after gradient-based optimization, we observe a substantial improvement that eventually leads the duck to the target position. This confirms the usefulness of gradients computed in DiffPD using the penalty-based contact method.

Complementarity-based contact For the contact model described in the complementarity form, our backpropagation algorithm assumes the contact set is a small subset of full DoFs. Specifically, Alg. 5 requires a relatively small size of $\mathcal{C}$ at each time step to gain substantial speedup over directly solving the modified linear system without leveraging the low-rank update. Given that $\mathcal{C}$ is a subset of surface vertices, whose number is much fewer than the number of interior vertices in a typical 3D volumetric deformable body, such an assumption can be easily satisfied in many applications. Indeed, in the next section we will present various 3D examples involving contact, none of which have more than 6% active contact nodes throughout simulation.

The assumption that $|\mathcal{C}|$ is relatively small is much more likely to be violated when we simulate a co-dimensional object, e.g., a one-dimensional rope or a piece of cloth in 3D, in which case it is entirely possible to have all nodes in $\mathcal{C}$ at some point. Although simulating co-dimensional objects is beyond the scope of this work, it can be a good test to reveal a critical ratio where the speedup from Alg. 5 starts to diminish. To mimic a co-dimensional object, we engineer a "Napkin" example consisting of one-layer voxels (Fig. 4-8) falling onto a spherical obstacle with an adjustable solid angle to control the size of $|\mathcal{C}|$. The relative size of $|\mathcal{C}|$ is capped by 50% when all the bottom nodes are in contact with the spherical obstacle (Fig. 4-8 right column). We vary the mesh resolution from $25 \times 25 \times 1$ voxels (4056 DoFs) to $100 \times 100 \times 1$ voxels (61206 DoFs) and report the running time of Newton's method and DiffPD in Table 4.1 for each resolution and contact set size. We can use Table 4.1 to decide between using our low-rank update method and directly solving the modified matrix in a downstream

Table 4.1: Average running time per step in the napkin example with various mesh resolution ("Res.") and relative contact set size from 6% to 50% (Fig. 4-8 top row). The reported time is averaged over all steps when the napkin is in contact with the obstacle. All times are in seconds. For each mesh resolution and each relative contact set, we report the running time from both Newton's method and DiffPD with the shorter time in bold.

Res.	Method	6%	24%	38%	50%
$25 \times 25 \times 1$ (4056 DoFs)	Newton-PCG	0.8	1.6	1.6	1.5
	DiffPD	**0.1**	**0.6**	**1.2**	**1.4**
$50 \times 50 \times 1$ (15606 DoFs)	Newton-PCG	5.4	8.4	**10.2**	**8.5**
	DiffPD	**1.2**	**6.0**	11.3	10.5
$75 \times 75 \times 1$ (34656 DoFs)	Newton-PCG	14.7	**25.1**	**25.2**	**22.5**
	DiffPD	**7.1**	29.0	42.8	48.1
$100 \times 100 \times 1$ (61206 DoFs)	Newton-PCG	44.6	**65.0**	**47.3**	**48.5**
	DiffPD	**32.0**	169.8	158.7	163.4

application. For example, for around 15k DoFs, Table 4.1 suggests that the low-rank update method is faster until the relative size of $\mathcal{C}$ reaches around 40%.

4.1.6 Applications

Table 4.2: The basic information of all examples in DiffPD.

Sec.	Task name	# of elements	# of DoFs	h (ms)	# of steps
4.1.5	Cantilever	2048	8019	10	25
	Rolling sphere	552	2469	5	100
4.1.6	Plant	3863	29763	10	200
	Bouncing ball	1288	9132	4	125
4.1.6	Bunny	1601	7062	1	100
	Routing tendon	512	2475	10	100
4.1.6	Torus	568	3204	4	400
	Quadruped	648	3180	10	100
	Cow	475	2488	1	600
4.1.6	Starfish	1492	7026	33.3	200
	Shark	2256	9921	33.3	200
4.1.6	Tennis balls	640	978	5.6	150

In this section, we show various tasks that can benefit from DiffPD and classify them into four categories: system identification, inverse design, trajectory optimization, and closed-loop control. Although prior efforts on differentiable simulators have

Table 4.3: The right five columns report whether the example in DiffPD has gravity as an external force, imposes Dirichlet boundary conditions on nodal positions, handles contact, requires hydrodynamical forces ("Hydro."), and has actuators ("Act.").

Sec.	Task name	Gravity	Dirichlet	Contact	Hydro.	Act.
4.1.5	Cantilever	✓	✓			✓
	Rolling sphere	✓		✓		
4.1.6	Plant		✓			
	Bouncing ball	✓		✓		
4.1.6	Bunny	✓		✓		
	Routing tendon	✓	✓			✓
4.1.6	Torus	✓		✓		✓
	Quadruped	✓		✓		✓
	Cow	✓		✓		✓
4.1.6	Starfish				✓	✓
	Shark				✓	✓
4.1.6	Tennis balls	✓		✓		

demonstrated their capabilities on almost all these examples, we highlight that DiffPD is able to achieve comparable results but reduce the time cost by almost an order of magnitude. We provide a summary of each example in Table 4.2 and Table 4.3. For examples with actuators, we implement the contractile fiber model as discussed in Min et al. [107].

Gradients from a differentiable simulator enable the usage of gradient-based numerical optimization techniques to improve the performance of a certain design. Regarding the optimization algorithm, we use L-BFGS in our examples by default unless otherwise specified. We report the time cost and the final loss after optimization in Table 4.4. For fair comparisons, we use the same initial guess and termination conditions when running L-BFGS with different simulation methods. When reporting the loss in Table 4.4, we linearly normalize it so that after shifting and rescaling, a loss of 1 represents the average performance from 16 randomly sampled solutions and a loss of 0 maps to a desired solution. For examples using a bounded loss, we map zero loss to an oracle solution that achieves the lower bound of the loss (typically 0). For unbounded losses used in the walking and swimming robots (Sec. 4.1.6 and 4.1.6), we map zero loss to the performance of solutions obtained from DiffPD. We present more details about the implementation of each example as well as the optimization progress in our supplemental material.

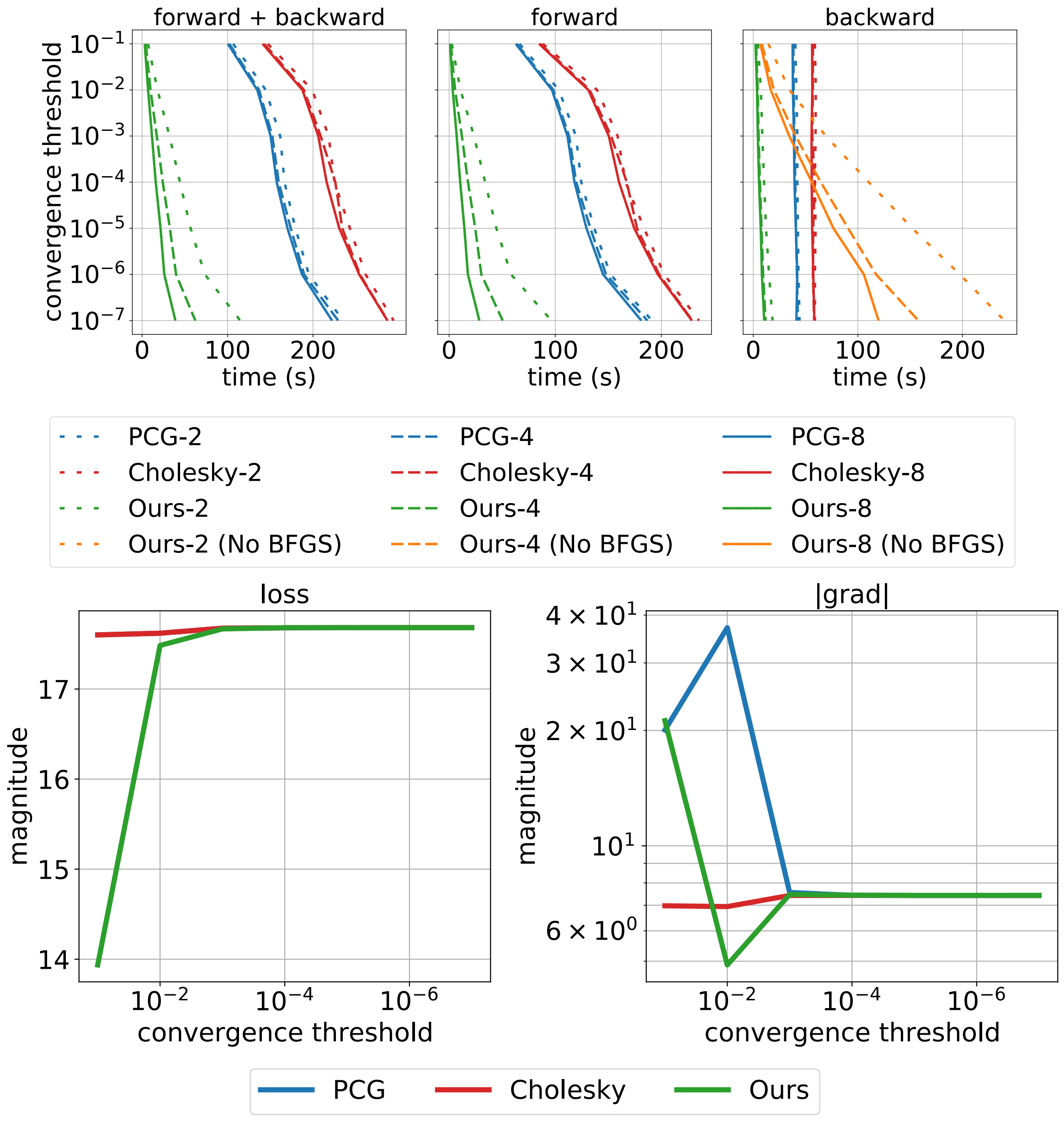


Figure 4-4: Top: the net wall-clock times (left), forward times (middle), and backpropagation times (right) for different convergence thresholds tested on the "Cantilever" example in Sec. 4.1.5. The results are obtained from simulating this example using each of the three methods: Newton-PCG (PCG), Newton-Cholesky (Cholesky), and DiffPD (Ours). The number following the method name denotes the number of threads used. Also shown are the results from running our method without applying the quasi-Newton method (orange, right). Bottom: the loss (left) and magnitude of the gradient of the loss (right) for different convergence thresholds used to terminate iterations in Newton's method and our simulator.

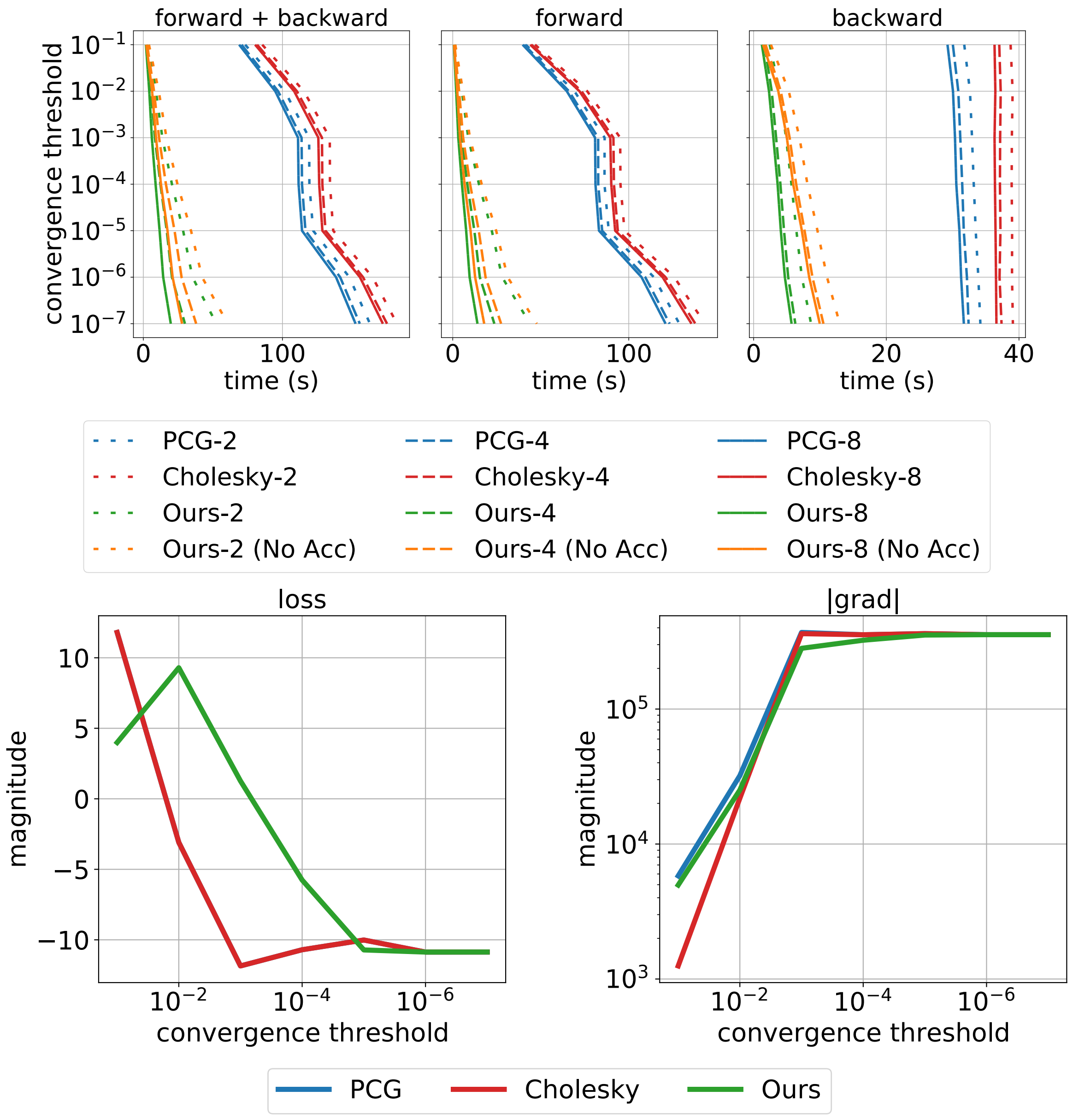


Figure 4-5: Top: the net wall-clock times (left), forward times (middle), and backpropagation times (right) for different convergence thresholds tested on the "Rolling sphere" example for contact handling. The results are obtained from three methods: Newton-PCG (PCG), Newton-Cholesky (Cholesky), and DiffPD (Ours). The number following the method name denotes the number of threads used. Also shown are the results from running our method without applying the further acceleration technique (Alg. 5) in Sec. 4.1.4 (orange). Bottom: the loss (left) and magnitude of the gradient of the loss (right) for different convergence thresholds used to terminate iterations in the Newton's method and our simulator.

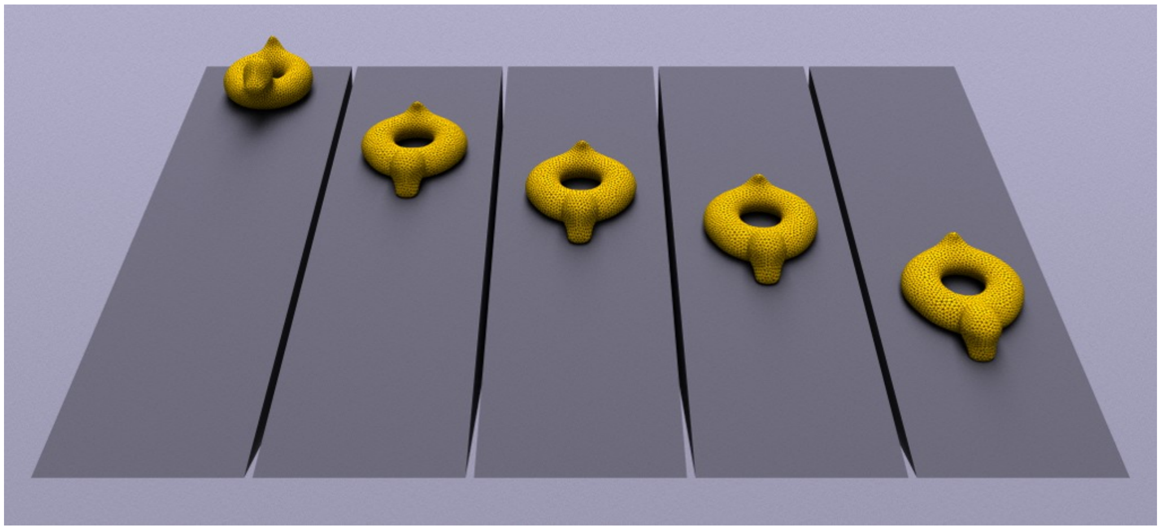


Figure 4-6: Slope. A rubber duck (16776 DoFs and 24875 tetrahedrons) slides on slopes with varying frictional coefficients implemented in DiffPD using a penalty-based contact and friction model [100]. Left: the initial position of the duck. Middle left to right: the final positions of the duck after two seconds with a decreasing frictional coefficient.

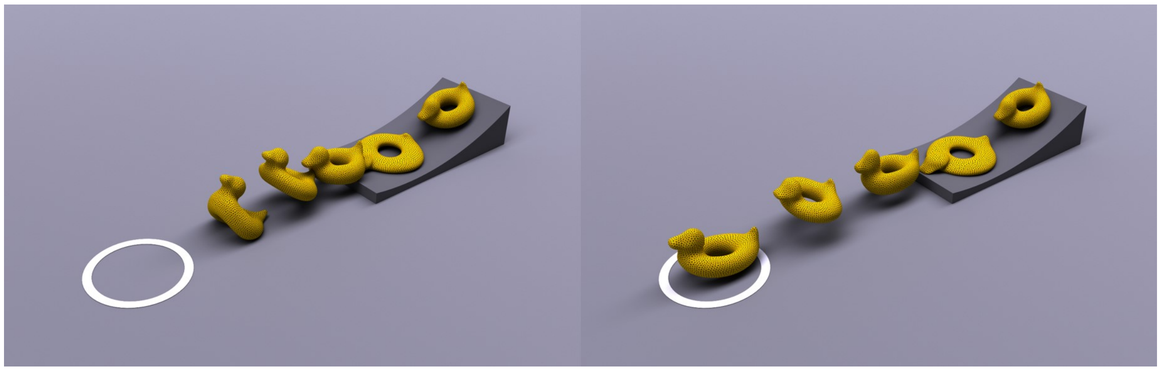


Figure 4-7: Duck. The same rubber duck in Fig. 4-6 now slides on a curved surface with trainable frictional coefficients. The goal in this test is to optimize the frictional coefficients so that the duck's final position after one second of simulation reaches the center of the white circle as closely as possible. We overlay the intermediate positions of the rubber duck at 0s, 0.25s, 0.5s, 0.75s, and 1s in simulation with an initial guess of the frictional coefficients before optimization (left) and the final coefficients after gradient-based optimization (right).

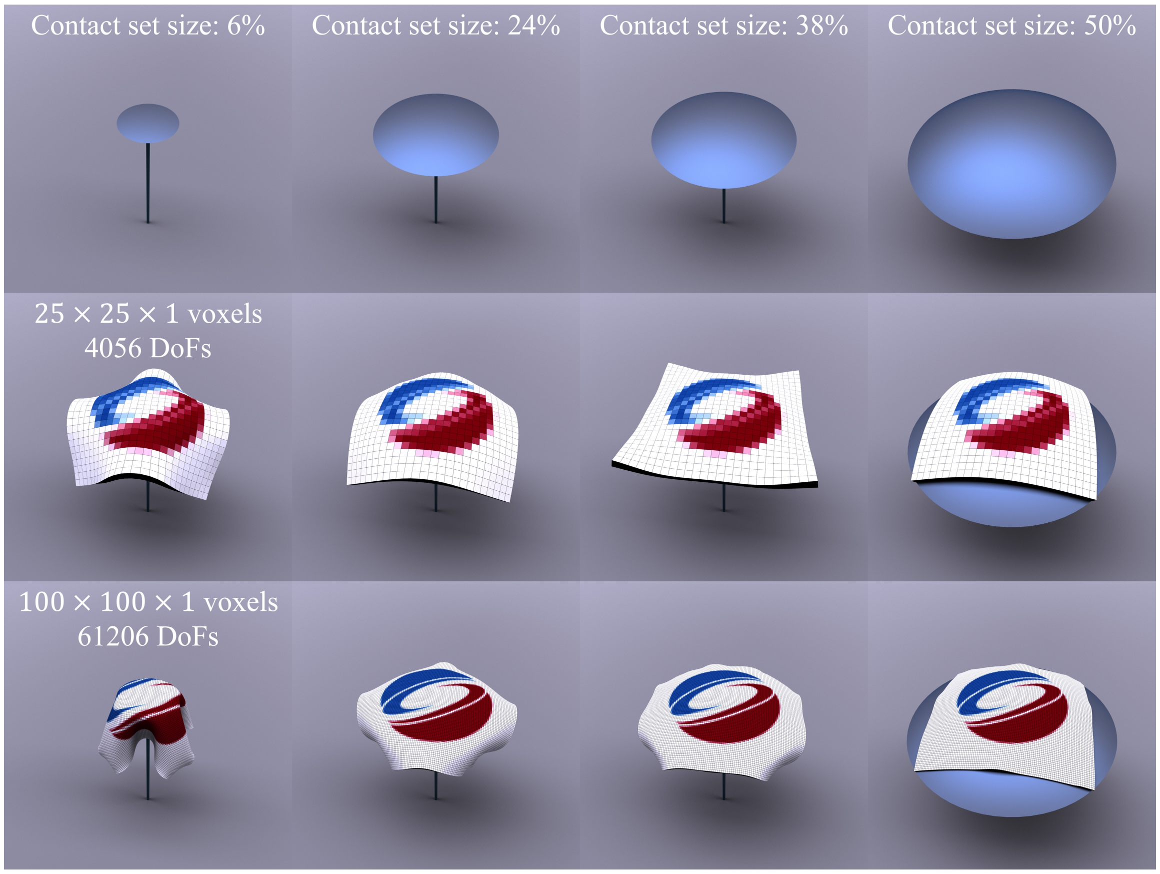


Figure 4-8: Ablation study on the relative size of the active contact set $|\mathcal{C}|$ and its influence on DiffPD's speed. We simulate a one-layer "Napkin" with resolutions from $25 \times 25 \times 1$ voxels (middle row) to $100 \times 100 \times 1$ voxels (bottom row) falling onto a spherical obstacle (blue, top row) with a varying solid angle. We report the relative size of $|\mathcal{C}|$ and the degrees of freedom of the napkin. Please refer to Table 4.1 for the detailed running time and our video for the full motion of the falling napkin.

Table 4.4: The performance of DiffPD on all examples. For each method and each example, we report the time cost of evaluating the loss function once ("Fwd.") and its gradients once ("Back.") in the unit of seconds. Also shown are the number of evaluations of the loss and its gradients ("Ev.") in BFGS optimization. The "Loss" column reported the normalized final loss after optimization (lower is better) with the best one in bold. Finally, we report the speedup computed as the ratio between the forward plus backward time of the Newton's method and of DiffPD, i.e., ratio between the sum of "Fwd." and "Back." columns. Note that no speedup is gained from DiffPD in the real-to-sim example "Tennis balls" because of its too small number of DoFs (Table 4.2).

Sec.	Task name	Newton-PCG				Newton-Cholesky				DiffPD (Ours)				
		Fwd.	Back.	Ev.	Loss	Fwd.	Back.	Ev.	Loss	Fwd.	Back.	Ev.	Loss	Speed
4.1.5	Cantilever	118.2	39.4	-	-	160.1	55.9	-	-	10.5	5.5	-	-	10×
	Rolling sphere	107.3	31.3	-	-	135.6	36.6	-	-	14.0	5.7	-	-	8×
4.1.6	Plant	1089.5	530.5	10	1.9e-3	929.6	525.2	10	1.9e-3	71.6	94.7	28	**5.9e-7**	9×
	Bouncing ball	269.3	90.9	43	**7.9e-2**	262.6	102.5	22	8.4e-2	15.8	14.2	12	9.6e-2	12×
4.1.6	Bunny	277.7	88.0	21	7.0e-3	358.2	126.9	29	**5.1e-3**	24.0	17.3	11	2.3e-2	9×
	Routing tendon	108.2	56.7	36	6.0e-4	107.3	58.7	38	**4.9e-4**	8.3	9.9	30	9.6e-4	9×
	Torus	751.9	210.3	47	-2.3e-3	719.9	212.4	43	**-2.4e-2**	84.3	81.9	27	0	6×
4.1.6	Quadruped	289.2	51.5	69	**-1.8e0**	246.3	47.8	54	-1.1e0	50.2	15.8	30	0	4×
	Cow	771.7	141.7	14	9.7e-1	620.1	140.2	20	9.8e-1	105.3	43.7	31	**0**	5×
4.1.6	Starfish	217.7	105.1	100	4.8e-1	244.0	129.4	100	1.4e-1	5.7	10.8	100	**0**	19×
	Shark	260.7	159.3	100	9.8e-1	599.4	241.8	100	**-9.0e-3**	35.5	15.3	100	0	8×
4.1.6	Tennis balls	54.6	6.4	14	7.2e-2	26.8	5.8	12	7.2e-2	24.1	15.9	41	**6.9e-2**	0.8×

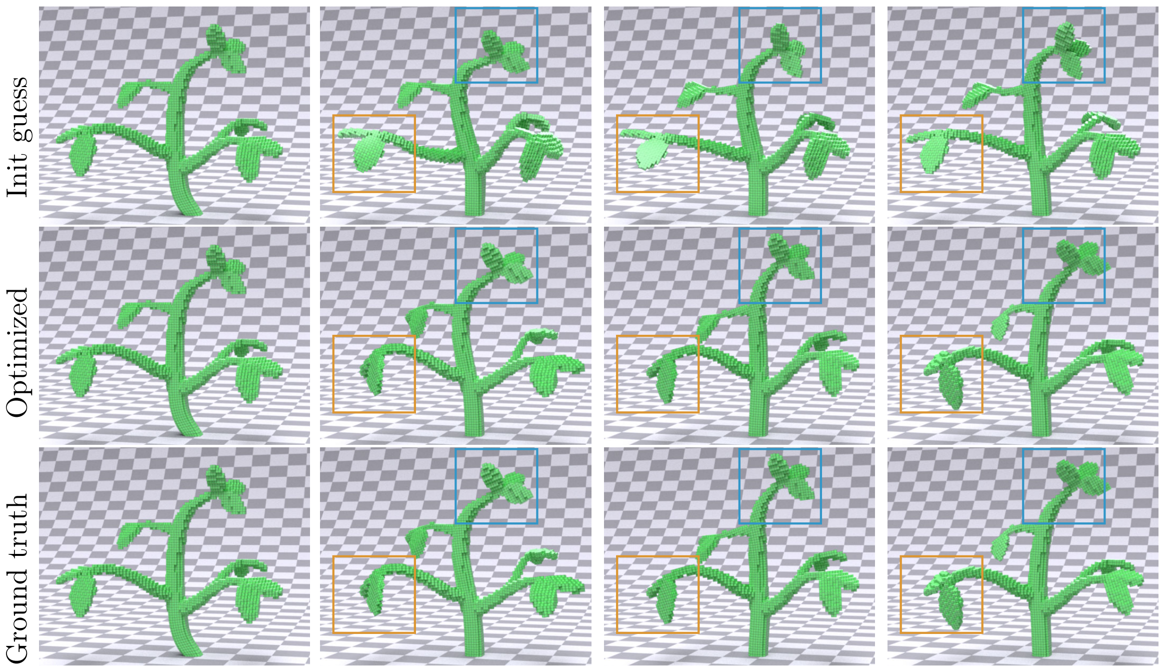


Figure 4-9: System identification Motion sequences of the "Plant" example sampled at the 1st, 100th, 150th, and 200th (final) frames (left to right). We generated three motion sequences with a random initial guess of the material parameters (top row), optimized material parameters (middle row), and the ground truth (bottom row). The colored boxes highlight the motion differences before and after optimization. The goal is to optimize the material parameters so that the motion of the plant matches the ground truth.

System Identification

In this section, we discuss two examples that aim to estimate the material parameters (Young's modulus and Poisson's ratio) from dynamic motions of soft bodies: the "Plant" example estimates material parameters from its vibrations, and the "Bouncing ball" example predicts its parameters from its interaction with the ground. We generate the ground truth using our forward PD simulator with a set of predefined material parameters.

Plant We first initialize an elastic, 3D house plant model with 3863 hexahedral elements and 29763 DoFs (Fig. 4-9). We impose Dirichlet boundary conditions at the root of the plant such that it is fixed to the ground. We apply an initial horizontal force at the start of simulation, causing the plant to oscillate. Starting from an initial guess using randomly picked material parameters, we deform a new plant in the same manner as the ground truth and optimize the logarithm of the Young's modulus and Poisson's ratio of the new plant to match that of the ground truth plant. The loss at each time step is determined as the squared sum of the element-wise difference in positions between the new plant and the reference plant.

After optimization, DiffPD, Newton-PCG, and Newton-Cholesky converge to local minima with a final Young's modulus of 1.00 MPa, 0.96 MPa, and 0.96 MPa respectively. Regarding the Poisson's ratio, DiffPD converge to 0.4 while both Newton's methods converged to 0.44. The reference plant is initialized with a Young's modulus of 1 MPa and a Poisson's ratio of 0.4. While the three methods all reach solutions that are similar to the ground truth, the optimization process is highly expedited by a factor of 9 for loss and gradient evaluation using our method (Table 4.4). We observe that DiffPD converged to a solution closer to the ground truth but used more function evaluations due to the numerical difference between DiffPD and the Newton's method. However, if DiffPD terminated after the same number of function evaluations (10) as the Newton's method, the optimized Young's modulus and Poisson's ratio would be almost identical to results from the Newton's method (0.97 MPa for Young's modulus and 0.44 for Poisson's ratio), implying the 9× speedup indeed comes from DiffPD's improvements over the Newton's method on the simulation side.

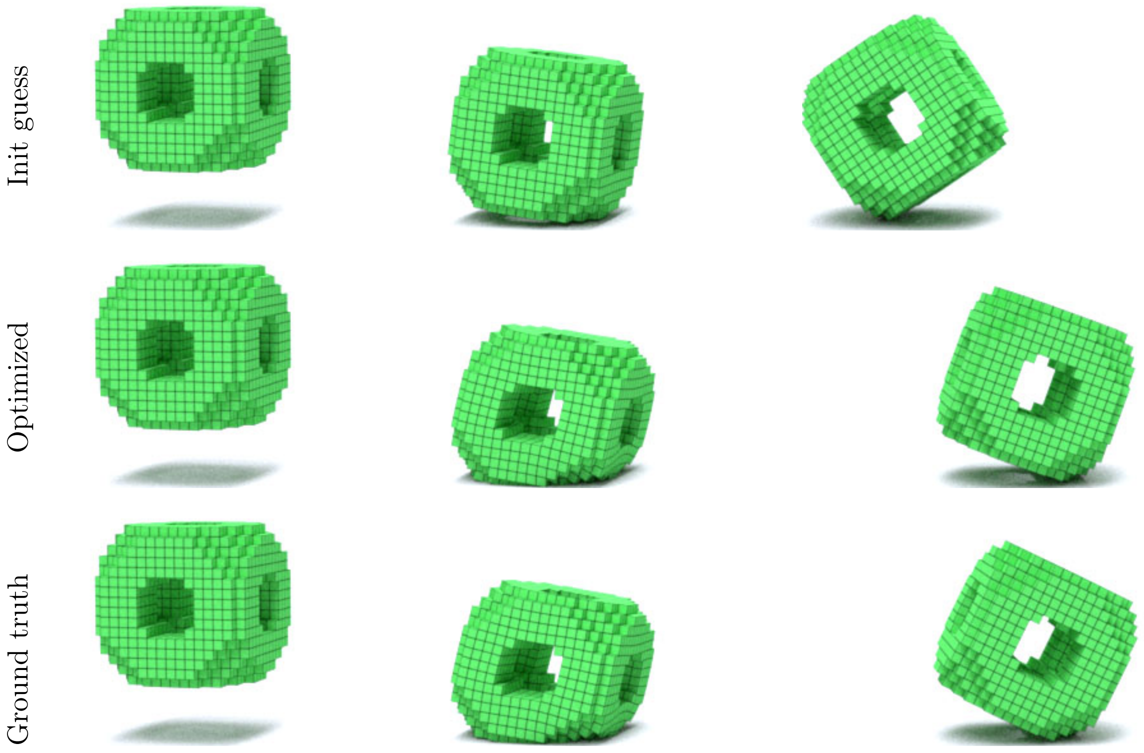


Figure 4-10: System identification Motion sequences of the "Bouncing ball" example sampled at the 1st frame (left), the 19th frame when collision occurs (middle), and the 125th (final) frame (right). We generated three motion sequences with a random initial guess of the material parameters (top row), optimized material parameters (middle row), and the ground truth (bottom row). The goal is to optimize the material parameters so that the motion of the ball matches the ground truth.

Bouncing ball In this example, we consider a ball with 1288 hexahedral elements and 9132 DoFs thrown at the ground from a known initial position (Fig. 4-10). The ball has three cylindrical holes extruded through the faces in order to produce more complex deformation behavior than a fully solid ball. This example uses the complementarity-based contact model in Sec. 4.1.4. We can estimate the material parameters of a bouncing ball by observing its behavior after it collides the ground. The loss definition is the same as in the parameter estimation of the "Plant" example. Regarding the optimization process, all three methods converged to a Young's modulus of 1.78 MPa and Poisson's ratio of 0.2. The ground truth values for the Young's modulus and Poisson's ratio are 2 MPa and 0.4 respectively. While the optimized material parameters are significantly different from the ground truth values, the motion sequences are very similar as reflected by the final loss in Table 4.4 and Fig. 4-10. Since the loss function is defined on the motion only, there could exist many material parameters that result in close-to-zero loss. As in the "Plant" example, our method enjoys a substantial speedup (12×) in computation time even with the inclusion of collisions in simulation.

Initial State Optimization

We present two examples demonstrating the power of using gradient information to optimize the initial configuration of a soft-body task. In the "Bunny" example, we optimize the initial position and velocity of a soft Stanford bunny so that its bounce trajectory ends at a target position. In the "Routing tendon" example, we optimize a constant actuation signals applied to each muscle in a soft cuboid with one face sticky on the ground so that the corner at the opposite face reaches a target point at the end of simulation.

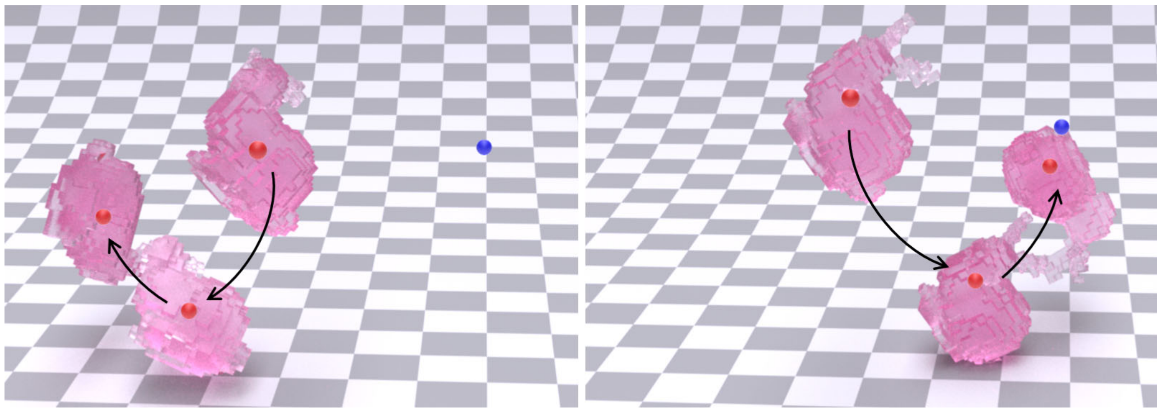


Figure 4-11: Inverse design Initial (left) and optimized (right) trajectory of the "Bunny" example. The red dots indicate the location of the center of mass of the bunny, and the blue dot is the target location. The goal is to adjust the initial position, velocity, and orientation of the bunny so that its final center of mass can reach the target location.

Bunny For this example, we optimize the initial pose and velocity of a Stanford bunny (1601 elements and 7062 DoFs) so that its center of mass (red dots in Fig. 4-11) reaches a target position (blue dot in Fig. 4-11) when the simulation finishes. This example uses the complementarity-based contact model, and we add 251 surface vertices (753 DoFs) to the set of possible contact nodes – approximately 10.7% of the 2354 vertices. Fig. 4-11 illustrates the trajectory of the bunny before and after optimization: the initial guess generates a trajectory almost to the opposite direction of the target, and the optimized trajectory ends much closer to the target. Note that none of the three methods solve this task perfectly: the trajectory does not reach the target even after optimization. This is because the target is chosen arbitrarily rather than generated from simulating a ground truth bounce trajectory, so it is not guaranteed to be reachable. According to Table 4.4, the final loss from DiffPD is larger than from the Newton's method (the center of mass is at a distance of around 7 times the length of one element from the target position as opposed to roughly 4 times). However, the increase in performance more than makes up for it. Using 8 threads, our method achieves a speedup of 9 times overall with a large set of potential contact points.

Routing tendon We initialize a soft cuboid with 512 elements and 2475 DoFs and impose Dirichlet boundary conditions such that its bottom face is stuck to the ground. We also add actuators to each element and group them into 16 muscle groups. The level of actuator activation is a scalar between 0 to 1, indicating muscle contraction and expansion, respectively. The elements within a specific actuation group all share the same, time-invariant actuation signal to be optimized in order to manipulate the endpoint (red dot in Fig. 4-12) of the soft body to reach a target point (blue dot in Fig. 4-12). The normalized losses at the final iteration for each of the methods (Table 4.4) are all close to zero, indicating that the task is solved almost perfectly. However, using DiffPD, we observe a 9× speedup over the Newton's methods.

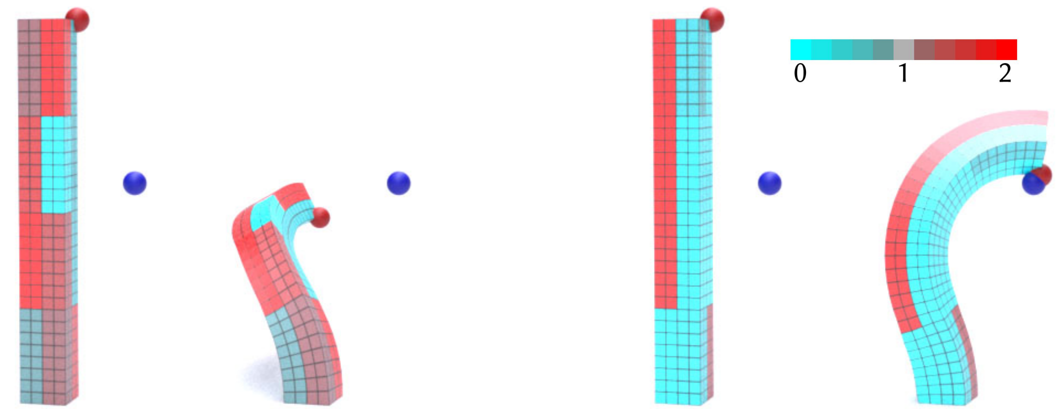


Figure 4-12: Inverse design The initial and final states of the "Routing tendon" example before and after optimization. The goal is to let the end effector (red dot) hit a target position (blue dot) at the end of simulation. Left: the initial configuration of the tendon with randomly generated actuation signals. The red (muscle expansion) and cyan (muscle contraction) colors indicate the magnitude of the action. Middle left: final state of the tendon with random actuation. Middle right: initial configuration of the tendon with optimized actuation. Right: final state with optimized actuation.

Trajectory Optimization

In order to demonstrate the applicability of our system's differentiability to solving complex trajectory optimization tasks, we apply our simulator to three locomotion tasks: a "Torus", a "Quadruped", and a "Cow". All three robots are equipped with muscle fibers whose sequences of actions are to be optimized, and the goal for all three robots is to walk forward without losing balance or drifting sideways. All examples use the complementarity-based contact model in Sec. 4.1.4.

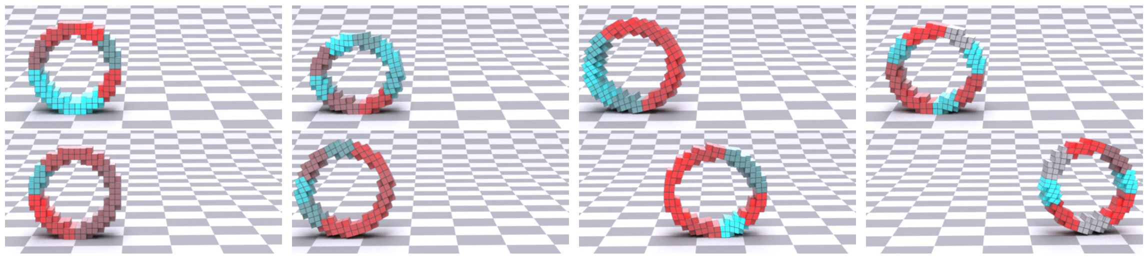


Figure 4-13: Trajectory optimization The motion sequence of the "Torus" example with random actions (top) and after optimizing the action sequences (160 parameters to be optimized) with DiffPD (bottom). The goal is to maximize the rolling distance of the torus while maintaining its balance. The red and cyan color indicates the magnitude of the action signal. In particular, the expansion and contraction pattern (middle left and middle right) allows the torus to roll forward.

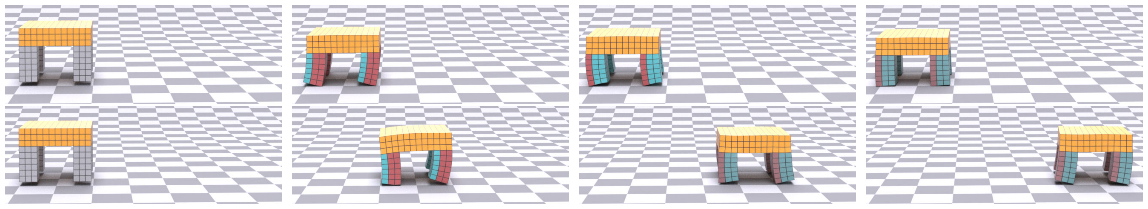


Figure 4-14: Trajectory optimization The motion sequences of the "Quadruped" example with sinusoidal control signals whose 3 parameters are to be optimized. The goal is to maximize the walking distance of the quadruped. Top: the motion sequence with a random sinusoidal wave of actions. Bottom: the motion sequence after optimization with DiffPD.

Torus In our first trajectory optimization example, a torus is tasked with rolling forward as far as possible in 1.6 seconds, simulated as 400 steps of 4 milliseconds in length (Fig. 4-13). To achieve this, we set the objective to be the negation of the robot's center of mass in x at the final step of its trajectory. Eight muscle tendons are routed circumferentially along the center of the torus, combined, creating a circle that can be actuated along any of eight segments. The optimization variables are the actuation of the each muscle at each of 20 linearly spaced knot points, and the actual action sequences are generated by linearly interpolating variables at these knot points. Since there are 8 muscles, this results in 160 decision variables overall. Since this example requires frequent contacts, we use a convergence threshold of 1e-6 as indicated by the evaluation experiment in Sec. 4.1.5.

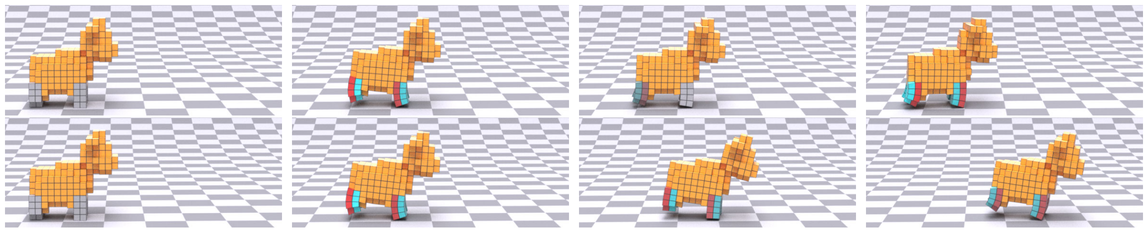


Figure 4-15: Trajectory optimization The motion sequences of the "Cow" example with sinusoidal control signals whose 3 parameters are to be optimized. The goal is to maximize the walking distance of the cow. Top: the motion sequence with a random sinusoidal wave of actions. Bottom: the motion sequence after optimization with DiffPD.

The major challenge in optimizing the sequence of actions of this rolling torus lies in the fact that it constantly breaks and reestablishes contact with the ground. When running L-BFGS on this example, we noticed more local minima than previous examples and L-BFGS often terminated prematurely without making significant progress. To alleviate this issue, we randomly sampled 16 initial solutions and selected the best among them to initialize L-BFGS optimization, which eventually learned a peristaltic contraction pattern that allows it to start rolling forward and make considerable forward progress (Fig. 4-13); further, DiffPD provides a 6× speedup using 8 threads compared to the Newton's method.

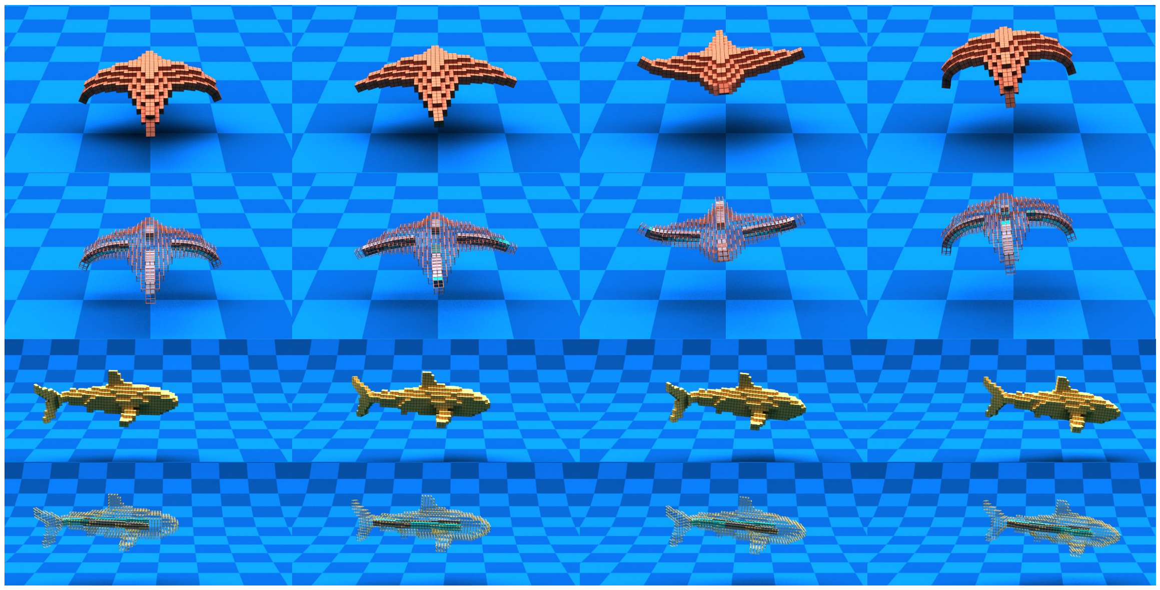


Figure 4-16: Control optimization Motion sequences from the optimized closed-loop, neural network controllers of "Starfish" (first row) and "Shark" (third row), with the corresponding muscle fibers plotted in the second and fourth rows. The goal is to optimize a controller so that the marine creatures can rise ("Starfish") or swim forward ("Shark"). The gray and cyan colors on the surface of the muscle fibers indicate the magnitude of the actuation, with gray being zero actuation and blue the maximum contraction.

Quadruped For our second trajectory optimization example, a quadruped is tasked with moving forward as far as possible in 1 second. The same performance objective is applied here as in the "Torus" example, however in this example a simpler control scheme is implemented. This robot has eight muscles, routed vertically along the front and back face of each leg, allowing the legs to bend forward or backward. For each leg, the front and back muscle groups are paired antagonistically, however they are allowed a different maximum actuation strength – a parameter to be optimized. Finally, the entire quadruped is provided a single sinusoidal control signal, whose frequency is to be optimized, that actuates each leg synchronously. These front and back actuation strengths, combined with the frequency of the input signal, provide 3 parameters to be optimized. After optimization, the quadruped was able to walk forward several body lengths (Fig. 4-14). In terms of the speedup, DiffPD accelerates loss and gradient evaluation by a factor of 4 compared to the Newton's method.

Cow For our third and final trajectory optimization example, a cow quadruped based off Spot [33] is tasked with walking forward as far as possible in 0.6 seconds (Fig. 4-15). This is a particularly difficult task, as Spot's oversized head makes it front-heavy, and prone to falling forward. In order to compensate, we regularized the objective to promote a more upright gait, adding an additional -0.3 times the center of mass in z to regularize the forward objective. Spot uses the same controller and muscle arrangement as the "Quadruped" example, and a convergence threshold of 1e-6 is used during optimization. Similar to previous examples, the cow optimizes to walk forward and DiffPD provides a 5 times speedup compared to the Newton's method.

Discussion Locomotion tasks generally involve significant contact, which limits the speedups (4-6 times in the examples above) DiffPD is able to achieve compared to contact-free problems. Given the complexity of planning the motion of walking robots with contacts (optimizing each of the three examples above took hours to converge with the Newton's method), a 4-6 times speedup is still favorable. It is also worth noting that for the "Quadruped" and "Cow" examples, optimization with the Newton's method led to solutions significantly different from DiffPD, as indicated by the final loss reported in Table 4.4. We believe this is mostly due to the algorithmic difference between the Newton's method and DiffPD: as discussed in Liu et al. [96] and Sec. 4.1.3, DiffPD is essentially running the quasi-Newton method (as opposed to the Newton's method) to minimize the objective in Eqn. (4.5) which is typically not convex. Therefore, multiple critical points may exist especially when contacts are involved. For the three locomotion tasks in this section, it is possible that DiffPD and two Newton's methods each explored different critical points individually and led to different solutions.

Closed-Loop Control

Finally, inspired by Min et al. [107], we consider designing a closed-loop neural network controller for two marine creatures: "Starfish" and "Shark" (Fig. 4-16). For each example, we specify muscle fibers as internal actuators similar to Min et al. [107] in the arms of the starfish and the caudal fin of the shark. We manually place velocity sensors on the body of each example serving as the input to the neural network controller. The goal of these examples is to optimize a swimming controller so that each fish can advance without drifting sideways. To achieve this, we define the loss function as a weighted sum of forward velocities and linear velocities at each time step. In terms of the neural network design, we choose a 3-layer multilayer perceptron network with 64 neurons in each layer (30788 parameters in "Starfish" and 22529 parameters in "Shark"). We use the hyperbolic tangent function as the activation function in the neural network. Unlike prior examples for which L-BFGS is used for optimization, we follow the common practice of using gradient descent with Adam [78] to optimize the neural network parameters. During optimization, we use a convergence threshold of 1e-3 in DiffPD and the Newton's method. Table 4.4 provides the final loss after optimization, and we observe a speedup of 8-19 times for the two examples respectively.

Comparisons to reinforcement learning We compare our gradient-based optimization method to PPO [126], a state-of-the-art reinforcement learning algorithm. In particular, we use the forward simulation of DiffPD as the simulation environment for PPO. For a fair comparison, we construct and initialize the network for both DiffPD and PPO with the same random seed. We also implement code-level optimization techniques as suggested in Engstrom et al. [47] and tune PPO hyperparameters towards its best performance. Please refer to our supplemental material for more implementation details.

When comparing the performance of PPO to gradient-based algorithms like Adam or L-BFGS, we expect gradient-based optimization to be more sampling efficiency than PPO as gradients expose more information about the soft body dynamics that are not accessible to PPO. Note that this does not ensure gradient-based methods are always faster than PPO when measured by their wall-clock time, because each sample in a gradient-based method requires additional gradient computation time. Furthermore, the sampling scheme in PPO is massively parallelizable. We report the optimization progress of PPO and our method in Fig. 4-17. Note that unlike other examples, we follow the convention in reinforcement learning of maximizing a reward as opposed to minimizing a loss. In particular, a zero reward indicates the average performance of randomly selected unoptimized neural networks, and a unit reward is the result from DiffPD after optimization. We conclude from Fig. 4-17 that Adam and DiffPD achieves comparable results to PPO but is more sampling efficient by one or two orders of magnitude. Regarding the wall-clock time, we observe a speedup of 9-11 times for both examples respectively, although each sample in DiffPD is more expensive due to the gradient computation.

A Real-to-Sim Experiment

We end this section with a real-to-sim experiment "Tennis balls" to highlight the value of DiffPD in potential real-world applications. In this example, we capture

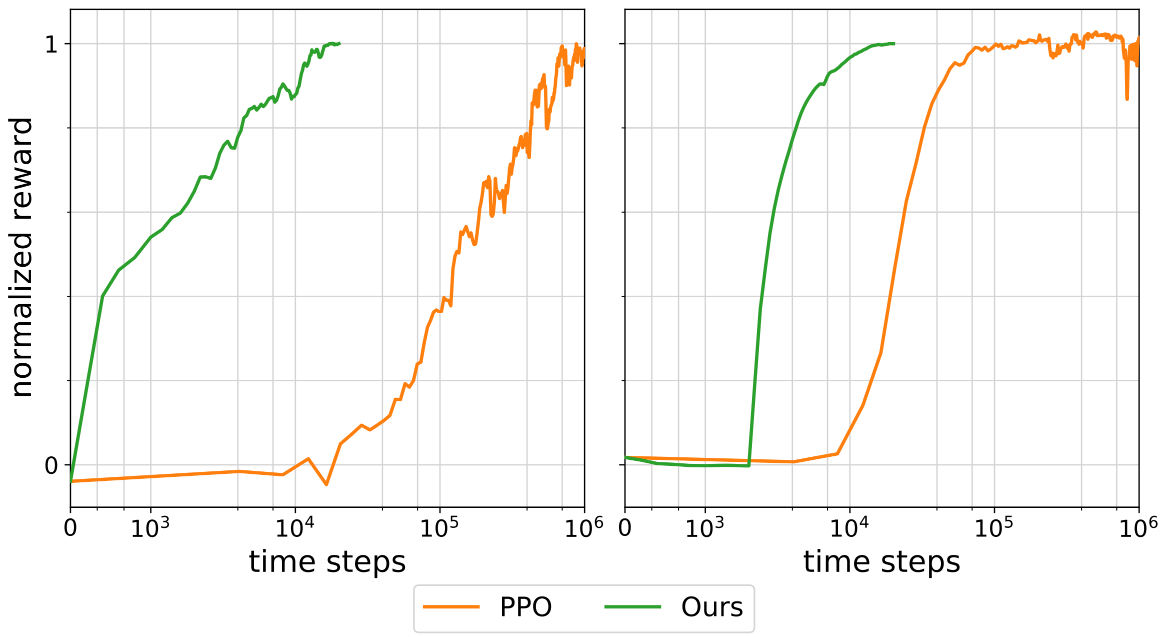


Figure 4-17: The optimization progress of Adam plus DiffPD (green) and PPO (orange) for "Starfish" (left) and "Shark" (right). Note that the axis of time steps spent during training or optimization is plotted on a logarithmic scale.

a video clip of two colliding tennis balls on a flat terrain, from which we aim to estimate the camera information, the initial position and velocity of each ball, and the parameters in the contact model. We model each ball in simulation using a mesh of sphere with 320 tetrahedrons and 489 DoFs (Table 4.2). We model the ball-ground contact with the complementarity-based method and use the following penalty-based model to compute the ball-ball contact: when the two balls are in contact, we add a restitution force computed as the product of a stiffness parameter to be optimized and the difference between the ball diameter and the actual distance between the two ball centers. Essentially, the restitution force can be treated as a spring model with a rest length equal to the ball diameter. Additionally, we add a frictional force whose direction is orthogonal to the restitution force and whose magnitude is controlled by a frictional coefficient to be optimized.

To define a loss function that measures the discrepancy between the simulated and actual motions of the two balls, we first extract the pixel location of two balls' centers in each frame of the video clip. Next, we compute in simulation the position of each ball and project them to the same image space through a pinhole camera model. We define the loss function as the difference between the pixel locations of the balls in simulation and from the video clip. By minimizing this loss, we get our estimation of the camera information, the initial state of each ball, and the parameters in the contact model.

We summarize our optimization results in Table 4.4 and Fig. 4-18. We randomly sample multiple sets of parameters and pick those with the smallest loss as the initial guess to our optimization (Fig. 4-18 left), which shows motion sequences similar to those in the video clip but still with substantial visual differences. The optimization process refines our estimation of the parameters and manages to further suppress the loss and mimics the motions in the video more closely (Fig. 4-18 middle), indicating that the simplified material and contact model can still be useful for real-world applications. The results can be further improved if we take into account camera lens distortion or replace the penalty-based collision model between two balls with a more accurate one, which we leave as future work.

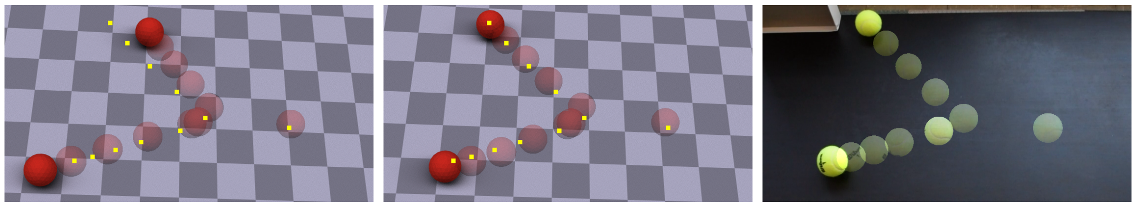


Figure 4-18: A Sim-to-Real Experiment Motion sequences of the "Tennis balls" example before (left) and after optimization (middle). The corresponding video clip is shown on the right. Transparent balls indicate the balls' intermediate locations. To visualize the difference between motion sequences in simulation and reality, we use yellow squares in the rendered images (left and middle) to denote the corresponding pixel locations of the balls' centers from the video clip.

4.1.7 Discussion and Limitations

Differentiable soft-body simulation with proper contact handling is a challenging problem due to its large number of DoFs and complexity in resolving contact forces. We believe one ambitious direction along this line of research is to provide a physically realistic simulator that can facilitate the design and control optimization of *real* soft robots. To close the simulation-to-reality gap, some nontrivial but rewarding enhancements need to be integrated into our current implementation. First and foremost, similar to other PD papers, a major limitation in DiffPD is its assumption on the energy function of the material model. Even though technical solutions to supporting general hyperelastic materials in PD exist [96], it turns out that supporting such materials in DiffPD is not straightforward. This is because the derivation in Sec. 4.1.3 starts to fall apart from Eqns. (4.16) and (4.17) when hyperelastic material models are used, forcing DiffPD to reassemble the Hessian matrix $\nabla^2 E$ in each time step during backpropagation. Although we can still apply the iterative solver from Sec. 4.1.3 in this case, we no longer observe a speedup over a direct solver (Fig. 4-19). Therefore, we switched to the direct solver for backpropagation in DiffPD when hyperelastic materials are used and leave speeding it up as future work.

Second, our contact methods do not fully resolve differentiable, complementarity-based contact and friction. Due to the focus of this paper, our choice of the contact model was intentionally biased towards ensuring differentiability and compatibility with PD. It would be more accurate and useful to upgrade the contact models for both static and sliding frictional forces [97] or to apply a more realistic contact model [90] while maintaining its efficiency and differentiability in PD.

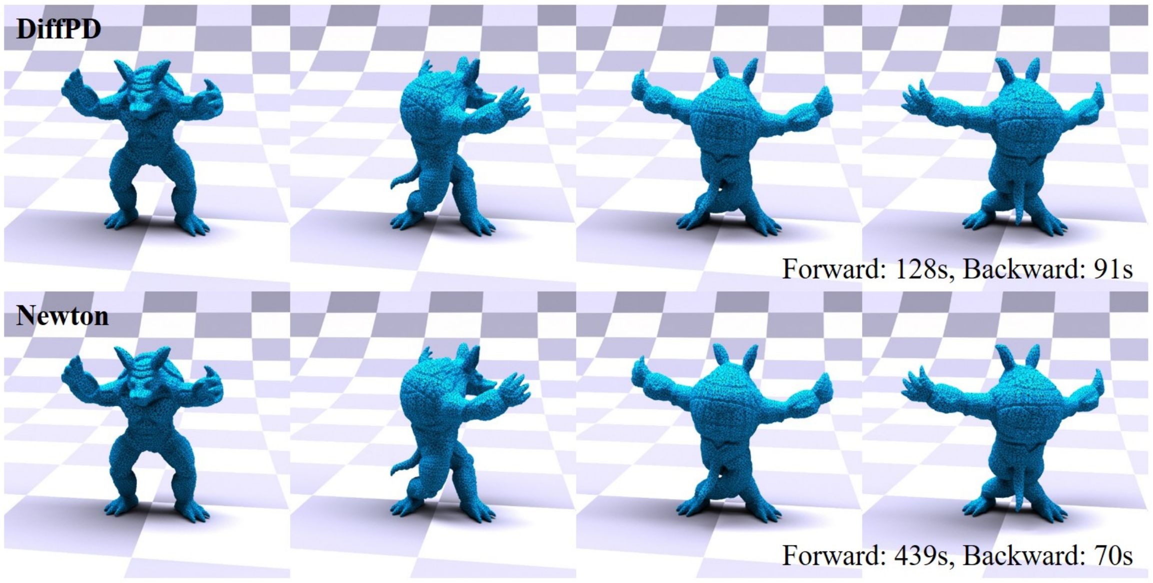


Figure 4-19: Twisting Armadillo (a 44337-DoF tetrahedral mesh) with the Neohookean material model for 1 second at 30 frames per second. We use DiffPD (top) and Newton's method (bottom) to compute the forward simulation and backpropagation. Left to right: the intermediate state of Armadillo at 0, 10, 20, and 30 frames. The visually identical motions from the top and bottom rows confirm the correctness of DiffPD's implementation of Neohookean material model. We report the time cost of DiffPD and Newton's method at the lower right corner of the images and no longer witness a speedup from DiffPD in backpropagation over a direct solver in Newton's method.

A third direction that is worth exploring is to improve the scalability of our algorithm. Currently, the largest example in this paper contains thousands of elements and tens of thousands of DoFs. It would be desirable to scale problems up by at least one or two orders of magnitude in order to explore the effects of more complex geometry. This would obviously be computationally expensive; it would therefore be interesting to explore possible GPU implementations of DiffPD method to unlock large-scale applications.

Fourth, although DiffPD is substantially faster than standard Newton's method when assumptions in PD hold, the speedup is less significant for locomotion tasks (4-6 times in our examples). We suspect it is the inclusion of contact that slows down DiffPD both in forward simulation and in backpropagation. Therefore, a more comprehensive analysis on the assumption of sparse contact in Sec. 4.1.4 would possibly reveal the source of inefficiency. Specifically, removing such an assumption would be much desired to unlock contact-rich applications, e.g., cloth simulation or manipulation.

Finally, there is room for improving the optimization strategies that can better leverage the benefits of gradients. In all our examples, we couple gradient information with gradient-based continuous optimization methods. Being inherently local, such methods inevitably suffer from terminating at local minima prematurely especially when the loss function has a non-convex landscape. It is worth exploring the field of global optimization methods or even combining ideas from gradient-free strategies, e.g., genetic algorithms or reinforcement learning, to present a more robust global optimization algorithm specialized for differentiable simulation.

4.2 Differentiable Fluid Simulation

The last section introduces a differentiable soft-body simulation with an in-depth discussion on its gradient derivation. Deriving gradients for a soft-body simulation is much more challenging than deriving gradients for the multicopter simulation in Chpt. 3, which is essentially a rigid-body simulator. This is because soft bodies have much more DoFs, and therefore computing gradients requires non-trivial numerical methods. The last section has shown that leveraging the numerical patterns in the governing equations of a soft-body system can speed up backpropagation substantially. This section will present another differentiable physics simulator with high DoFs, a simulator dedicated to Stokes flow simulation. We will show that backpropagation in this system is also a non-trivial task, which we will take care of with some insights from the numerical system.

4.2.1 Motivation

Fluidic devices are key building blocks for a variety of ubiquitous products, including medical diagnostic devices, filtration systems, bioreactors, internal combustion engines, hydraulic actuators, and even cooling manifolds for GPUs. However, designing complex fluidic devices is challenging as it requires expert knowledge and typically many trial-and-error iterations. These challenges promote the importance of finding computational strategies for simulating and designing these structures. Unfortunately, such approaches are challenging. Brute-force, high-resolution, physics-based simulations of fluidic systems are inherently slow and highly sensitive to geometric configurations and initial conditions, limiting progress in methods for computationally designing fluidic devices with high resolution and complex functions. Furthermore, performance-driven design methods (also often referred to as inverse methods) require using an expensive fluid simulation within a numerical optimization method. This effectively makes current approaches for performance-driven optimization impractical.

In this section, we present a first step toward functionally optimizing the design of fluidic devices, focusing on the more tractable *Stokes flow*, which is well-suited for the behaviors of desired fluidic functionality. Stokes flow assumes that fluid velocities are slow and fluid viscosity is relatively large (the Reynolds number $\mathrm{Re} \ll 1$). Additionally, in our approach, we use a parametric shape representation of a fluidic system – fluid-solid boundaries are represented using parametric surfaces. This has the advantage that the design process is intuitive for the designer (e.g., a designer specifies an initial shape). At the core of our approach is a differentiable Stokes flow simulator that efficiently solves not only the fluidic dynamics but also the gradients of the dynamics with respect to design parameters. This capability allows us to use this solver as a building block for gradient-based optimization algorithms when performance objectives (e.g., target fluid flows at inlets or outlets) are specified. Overall, our framework unlocks fast fluid flow simulation and gradient computation, making it amenable to continuous optimization.

Our proposed method shares similarities with topology or shape optimization, the two prominent techniques in engineering practice for functional design of fluidic devices [2]. The vast majority of prior methods focus on topology optimization for steady-state laminar flows paired with no-slip boundary conditions only [21, 58, 94, 13]. While topology optimization yields a geometrically expressive design space, this combination of rasterized and highly frictional boundaries limits both the realism and the *functional* expressiveness of the optimized designs. A less prevalent but more recent line of research is shape optimization of fluidic devices [150, 165], which is more related to our method. However, to our best knowledge, existing demonstrations from these papers are still coupled with no-slip boundary conditions only, and discussions on extensions to flexible boundary handling in shape optimization are sparse. Our method is in sharp contrast to prior as it simultaneously accommodates smooth parametric shape representations and handles explicit, versatile boundaries. We focus on spline-based parametric boundaries, which naturally yield smooth flows. Further, such parameterizations are low-dimensional (more tractable), more intuitive to reason about, and guarantee physically fabricable devices (i.e., no floating components) when compared with voxel-based parameterizations. Finally, with the careful treatment of sub-cell discretizations in our method, we support various boundary conditions (e.g., no-slip, traction, or no-separation boundary conditions) that allow the emergence of laminar flows in scenarios where such behavior would be anticipated.

To demonstrate the efficacy of our approach, we run performance-driven optimization for the design of complex 3D fluidic systems, including flow averagers, funnels, superposition gates, twisters, and switches. For each example, an engineer starts by specifying an initial fluid-solid boundary with splines, which are all easily parameterized with fewer than 50 degrees of freedom. The engineer then specifies a fluid flow at the inlets of the system and target fluid flow to be optimized at the system's outlets. For all examples, our approach manages to return an optimized design that significantly improves the performance of the device within less than 50 optimization iterations. Furthermore, we demonstrate in our fluidic switch example that our approach supports multifunctional design optimization over continuously varying input velocity configurations.

To summarize, this section contributes the following:

- A differentiable Stokes flow simulator with a continuous representation of the fluid-solid interface that naturally fits within an optimization framework;

- A sub-cell discretization paradigm in Stokes flow simulation that supports flexible boundary conditions, including no-slip, traction, and no-separation boundaries;

- An optimization pipeline for computational design of multifunctional fluidic devices with continuously varying input velocity configurations.

4.2.2 System Overview

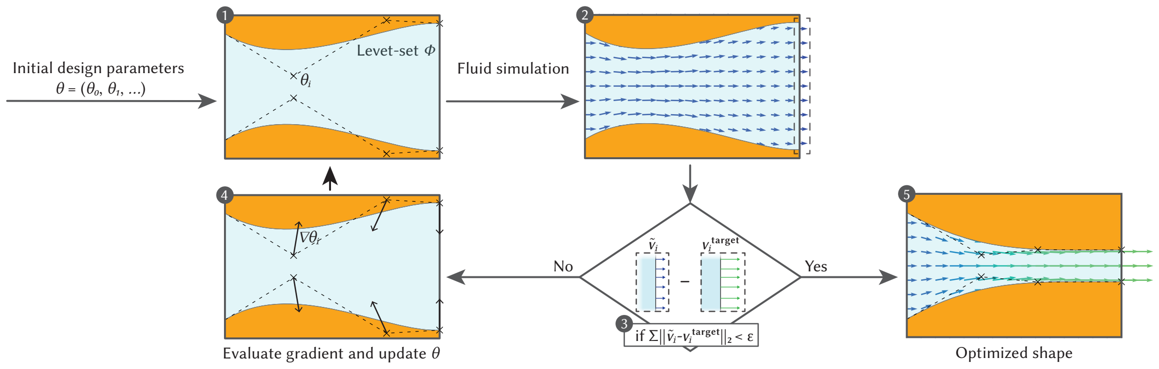

1
Initial design parameters
$\theta = (\theta_0, \theta_1, ...)$
Levet-set Φ
θ_i
Fluid simulation
2
No
Yes
$\tilde{v}_i$
v_i^{target}
3
if $\Sigma\|\tilde{v}_i - v_i^{\text{target}}\|_2 < \varepsilon$
4
$\nabla\theta_i$
Evaluate gradient and update θ
5
Optimized shape

Figure 4-20: An overview of our system: 1) The design parameter $\boldsymbol{\theta}$ (either explicitly given or randomly initialized) determines the fluid-solid boundaries in the design problem. 2) For any given $\boldsymbol{\theta}$, we simulate the Stokes flow in the fluidic domain and enforce different types of boundary conditions explicitly. 3) We then evaluate the loss function on the resulting velocity field and test it against the termination condition. 4) If the result is not optimal, we differentiate the loss with respect to $\boldsymbol{\theta}$ and its gradient is applied in a gradient-based local optimization method to update the design parameter. 5) The algorithm terminates with an optimized $\boldsymbol{\theta}$ and the corresponding design.

Our system is visualized in Fig. 4-20. As input, a user supplies a parameterized level-set geometry, for example, spline curves or NURBS surfaces. These manifolds separate the solid regions from regions with fluid flow. The user further specifies inlet flow velocities at some point on the fluid portion of the grid (fixed boundary conditions) and an objective to optimize. This objective could be, e.g., target flow velocities at certain designated outlet locations.

During optimization, the following quasi-Newton optimization loop is applied: the current design, as defined by the current parameter instantiation, is simulated on a regular grid with explicit handling of different types of boundary conditions. If the performance of the simulated device does not match the desired objectives of the user, the objective function is differentiated through the simulation with respect to the design parameters to produce a gradient, which is then used to improve the design. Otherwise, the optimization is terminated with a successful design.

4.2.3 Governing Equations

Since our work targets shape optimization of structures that modulate the flow properties of liquid media, we first present the mathematical model we have adopted for the governing equations of such fluid flows. Given their ubiquity in computational physics and computer graphics, established models of fluid flow such as the incompressible Euler equations for inviscid flows or, more generally, the Navier-Stokes equations for fluids with nontrivial viscosity [26] would be natural choices. However, given the difficulty of the continuous optimization in the inverse design problem at hand, we consciously restrict our material model to a narrower set of smoother, more well-behaved fluid behaviors. First, we specifically seek to model *steady-state flows* in order to avoid transient effects and avoid time dependencies in our optimization task. Second, in order to avoid local minima associated with non-unique solutions as well as boost the speed and conditioning of the optimization scheme, we employ the linearized form of the steady-state Navier-Stokes equations, also known as *Stokes flow* [85].

Incompressible Stokes equations We initially review the PDE form of the Stokes system and describe the boundary value problem that would be typically formulated for flow scenarios as in our target application. Let $\Omega \subset \mathcal{R}^d$ ($d = 2$ or 3) be a domain bounded by a smooth boundary Γ. In the standard Eulerian perspective, the Stokes equations yield a velocity field $\boldsymbol{v} : \Omega \to \mathcal{R}^d$ and a pressure field $p : \Omega \to \mathcal{R}$ as solutions to the PDE system:

$$-\eta \boldsymbol{\Delta v}(\boldsymbol{x}) + \nabla p(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Omega, \tag{4.44}$$

$$\nabla \cdot \boldsymbol{v}(\boldsymbol{x}) = 0 \qquad \boldsymbol{x} \in \Omega, \tag{4.45}$$

where η is the dynamic viscosity and $\boldsymbol{f}(\boldsymbol{x})$ an externally applied force field (e.g. gravity) if applicable. We note that Eqn. (4.44) is derived from the momentum equation $\nabla \cdot \boldsymbol{T}(\boldsymbol{x}) + \boldsymbol{f}(\boldsymbol{x}) = 0$ after substituting the linear constitutive law for the stress tensor $\boldsymbol{T}$:

$$\boldsymbol{D} = \frac{1}{2}[\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T], \tag{4.46}$$

$$\boldsymbol{T} = 2\eta \boldsymbol{D} - p\boldsymbol{I} = \eta[\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T] - p\boldsymbol{I}, \tag{4.47}$$

and using the incompressibility Eqn. (4.45) to simplify the result; here, $\nabla \boldsymbol{v}$ is the spatial gradient of $\boldsymbol{v}$, $\boldsymbol{D}$ the strain rate tensor, and $\boldsymbol{I}$ the $d \times d$ identity matrix.

Boundary conditions along the boundary Γ may be chosen from several types, according to the intended scenario and application. The most straightforward would be *Dirichlet boundary conditions*

$$\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{\alpha}(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Gamma_D \tag{4.48}$$

on any part of the boundary, denoted as Γ_D where we want to have a prescribed velocity profile $\boldsymbol{\alpha}(\boldsymbol{x})$, as in the inlet to the apparatus depicted in Fig. 4-21 (a). In those cases where we seek to model a highly viscous contact layer, a *no-slip* zero-Dirichlet boundary condition $\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{0}$ would also be enforced along the surface of the container wall; this is used in only a minority of our examples, but is certainly an option within our framework.

The remaining types of boundary conditions encountered in our framework involve the *traction* vector $\boldsymbol{\tau}(\boldsymbol{x}) = \boldsymbol{T}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x})$, defined on a boundary location $\boldsymbol{x} \in \Gamma$ with outward-pointing normal vector $n(\boldsymbol{x})$. A *traction condition*

$$\boldsymbol{\tau}(\boldsymbol{x}) = \boldsymbol{\beta}(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Gamma_T \tag{4.49}$$

would typically be used in outlets of our flow device where, instead of prescribing a flow profile, we would provide an externally applied force along the associated boundary (i.e. a cross-section of the fluid container) that intends to either impede or boost the flow. An example would be a permeable membrane affixed to an outlet that seeks to impede the flow by applying a resistive force. The specific type of boundary condition used in all our examples is $\boldsymbol{\tau}(\boldsymbol{x}) = \mathbf{0}$ (equivalently, $\boldsymbol{\beta}(\boldsymbol{x}) = \mathbf{0}$) which we would refer to as an *open boundary* condition and corresponds to the flow being allowed to transit through the domain boundary freely, without either being impeded or boosted by any external influence (see Fig. 4-21 (a)).

The final type of boundary condition we optionally employ in our framework is a *no-separation* (and, in essence also *no-friction*) boundary condition along the walls of the enclosing container. This mixed boundary condition is captured in the following equations:

$$\boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) = 0 \qquad \boldsymbol{x} \in \Gamma_S, \tag{4.50}$$

$$\boldsymbol{\tau}_t(\boldsymbol{x}) = \mathbf{0} \qquad \boldsymbol{x} \in \Gamma_S, \tag{4.51}$$

where the first component is conveyed by the scalar (1D) condition in Eqn. (4.50) and dictates that the flow should be parallel to the container wall (with $\boldsymbol{n}(\boldsymbol{x})$ being the normal vector at a boundary point $\boldsymbol{x} \in \Gamma_S$); this suggests that the flow will neither separate from the container, nor will it penetrate into it. Eqn. (4.51) dictates that the *tangential* component $\boldsymbol{\tau}_t$ of the traction vector should be equal to zero; this constraint has $(d-1)$ dimensions as it is projected on the tangential plane at each boundary location. Intuitively, this condition suggests that the fluid flow is not subject to any frictional forces that would impede its tangential motion; when combined with the no-separation condition this yields the same number of d equations per boundary point as in other types of boundary conditions. We employ this type of boundary condition broadly (albeit, not exclusively) in our examples, as it enables the emergence of the type of laminar steady-state flows that we would intuitively expect with a friction-free contact layer.

Relation to linear elasticity Well-known parallels exist between the Stokes problem and the PDEs of *linear elasticity*, which are broadly used in shape and topology optimization applications. We should emphasize that these analogies – stemming from the fact that both equations emerge from directly congruous conservation laws – are despite the fact that the underlying state variable has a different physical meaning for fluids versus elastic solids. In fluids, the PDE is defined over a *velocity* field, and in elastic media, over a *displacement* field.

Although we will demonstrate this analogy in its most stark form in the limit of *incompressible* linear elastic materials, we will start our review of this relation from the standard (i.e. compressible) linear elasticity PDE. For an elastic medium whose shape change is encoded via a deformation map $\boldsymbol{x}(\boldsymbol{X}) : \Omega \to \mathcal{R}^d$ (where $\boldsymbol{X}$ are material/undeformed coordinates and $\boldsymbol{x}$ the corresponding spatial/deformed locations), we define the *displacement* field as $\boldsymbol{u}(\boldsymbol{X}) = \boldsymbol{x}(\boldsymbol{X}) - \boldsymbol{X}$, and subsequently define the small-strain tensor $\boldsymbol{\epsilon}$ and Cauchy stress $\boldsymbol{\sigma}$ from a linear stress-strain relationship:

$$\boldsymbol{\epsilon} = \frac{1}{2}[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T], \tag{4.52}$$

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\epsilon} + \lambda \operatorname{tr}(\boldsymbol{\epsilon})\boldsymbol{I}, \tag{4.53}$$

where μ, λ here are the Lamé coefficients of the elastic material. Substituting the stress tensor $\boldsymbol{\sigma}(\boldsymbol{x})$ into the momentum equation $\nabla \cdot \boldsymbol{\sigma}(\boldsymbol{x}) + \boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$ (where $\boldsymbol{f}(\boldsymbol{x})$ are the external forces, if any) now yields the PDE of linear elasticity [131]:

$$-\mu\boldsymbol{\Delta}\boldsymbol{u}(\boldsymbol{x}) - (\mu + \lambda)\nabla[\nabla \cdot \boldsymbol{u}(\boldsymbol{x})] = \boldsymbol{f}(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Omega. \tag{4.54}$$

The relation to the Stokes equations will start becoming more apparent if we consider an almost incompressible material for which the value of λ is significantly larger than that of μ; although the solution of the PDE evolves smoothly and continuously as the parameter λ asymptotically reaches infinity, the exact form of the PDE in Eqn. (4.54) would not be the ideal way to express it, due to the unbounded coefficients involved. Instead, we can derive a better behaved, equivalent system in the spirit of mixed formulations [25, 167], by introducing a new, auxiliary state variable $r(\boldsymbol{x})$ defined as

$$r(\boldsymbol{x}) = -(\mu + \lambda)\nabla \cdot \boldsymbol{u}(\boldsymbol{x}). \tag{4.55}$$

By substituting this expression into Eqn. (4.54) and rearranging terms in Eqn. (4.55), we arrive at the following equivalent PDE system for compressible linear elasticity:

$$-\mu\boldsymbol{\Delta}\boldsymbol{u}(\boldsymbol{x}) + \nabla r(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Omega, \tag{4.56}$$

$$\nabla \cdot \boldsymbol{u}(\boldsymbol{x}) + \frac{1}{\mu + \lambda} r(\boldsymbol{x}) = 0 \qquad \boldsymbol{x} \in \Omega. \tag{4.57}$$

Once we have arrived at this form, the analogy between the Stokes equations and the above equations of linear elasticity start becoming more apparent. We highlight the following observations:

- It should be clarified that any similarities between the two governing laws are restricted to the form of their PDEs, while the underlying state variables are semantically distinct. Specifically, η, $\boldsymbol{v}$, and p in Eqns. (4.44, 4.45) play the same role as μ, $\boldsymbol{u}$, and r in Eqns. (4.56, 4.57), although their physical meanings are quite different, e.g., in Stokes flow, $\boldsymbol{v}$ is a velocity field, where in elasticity $\boldsymbol{u}$ refers to a field of elastic displacements. These semantic differences do not prevent us, however, from exploiting the similarities at the PDE level.

- It is known [25, 111] that the reformulated system in Eqn. (4.56) and (4.57) is smooth (and also, elliptic) and remains well behaved in the asymptotic limit $\lambda \to \infty$ when the coefficient of $r(\boldsymbol{x})$ in Eqn. (4.57) will merely vanish. The solution to the PDE system, itself, will smoothly and uniformly converge to a limit behavior as we asymptotically approach strict incompressibility.

- If we specifically consider the asymptotic case of strict incompressibility ($\lambda \to \infty$), then Eqn. (4.56) and (4.57) reduce *exactly* to the Stokes equations as stated in the prior paragraph.

The analogy (and, actually, equivalence) of the linear elasticity and Stokes PDEs would not be complete if we did not also address the form that the respective *boundary conditions* that the two sets of equations might employ. Dirichlet conditions, of course, are equally applicable to both formulations. Those boundary conditions, however, that involve the stress tensor $\boldsymbol{T}$ in Stokes flow and $\boldsymbol{\sigma}$ in linear elasticity require special attention. Taking the trace of Eqn. (4.52) yields $\mathrm{tr}(\boldsymbol{\epsilon}) = \nabla \cdot \boldsymbol{u}$; using this equality and the definition of r in Eqn. (4.55) allows us to rewrite the stress tensor from Eqn. (4.53) as

$$\begin{aligned}\boldsymbol{\sigma} &= \mu[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T] - \frac{\lambda}{\mu + \lambda} r \boldsymbol{I} \\ &= \mu[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T] - 2\nu r \boldsymbol{I}, \qquad (4.58)\end{aligned}$$

where $\nu = \frac{\lambda}{2(\mu+\lambda)}$ is the Poisson's ratio that approaches the value 0.5 in the incompressible limit. Once again, we observe that in the incompressible limit, the stress tensors in both linear elasticity and Stokes converge to the same limit form; as a consequence, so would any traction boundary conditions that would derive from this stress tensor. This demonstrates the asymptotic equivalence of Stokes and linear elasticity at the near-incompressible limit.

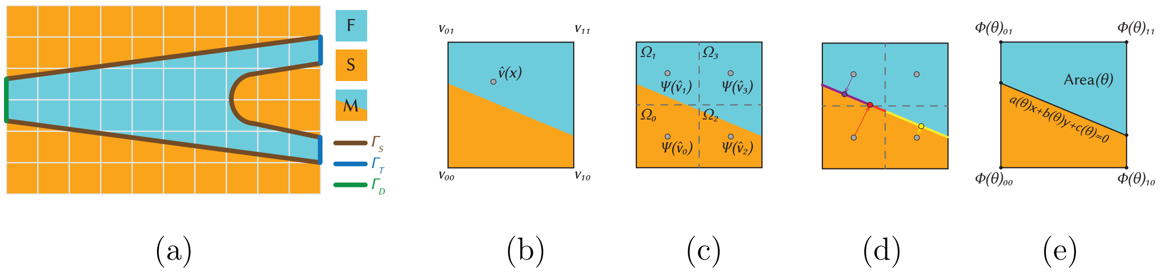


Figure 4-21: (a) the entire 2D space is discretized into fluid, solid, and mixed cells, with three types of boundaries as discussed in Sec. 4.2.3. (b) fluid velocities $\boldsymbol{v}_i$ are stored on grid nodes. (c) the fluid energy density is evaluated on different quadrature points and integrated over the entire cell by multiplying by the fluid occupied area at each subcell. (d) the Dirichlet boundary condition is enforced by integrating the values over the linearized interface, with the quadrature points obtained from the projection of quadrature points in (c) onto the interface. (e) all geometric information is defined by design parameters $\boldsymbol{\theta}$ and linearized within each subcell.

Our model: quasi-incompressible Stokes The aforementioned relation of Stokes and linear elasticity has previously been leveraged primarily to develop discretization and solution schemes for incompressible or near-incompressible elasticity that draw inspiration from established methods for Stokes [55, 168]. However, directly pursuing a discretization of the Stokes problem has its own subtleties; due to the incompressibility constraint, the associated discretizations – and especially variational formulations – take the form of saddle point problems, restricting somewhat the options for associated numerical solvers. Boundary treatment at sub-element precision is relatively nontrivial, especially if certain numerical properties of the discretization (e.g. symmetry) are to be preserved [167].

We have thus decided, in this initial venture into shape optimization involving fluid flows, to move in the opposite direction, and use the equations of linear elasticity in the *near-incompressible* (e.g. $\nu \approx 0.49$) but not strictly incompressible regime. We choose to use the common form of the linear elastic model in Eqn. (4.54) as opposed to the pressure-augmented system (Eqns. (4.56,4.57)), and also use the corresponding expression for the stress tensor, as in Eqn. (4.53) in the formulation of traction or no-separation/no-friction boundary conditions. The new governing equation for our quasi-incompressible Stokes flow model can be obtained by replacing μ and $\boldsymbol{u}$ in Eqn. (4.54) with the dynamic viscosity η and the velocity field $\boldsymbol{v}$ from Stokes flow:

$$-\eta\boldsymbol{\Delta v}(\boldsymbol{x}) - \frac{\eta}{1-2\nu}\nabla[\nabla \cdot \boldsymbol{v}(\boldsymbol{x})] = \boldsymbol{f}(\boldsymbol{x}) \qquad\qquad \boldsymbol{x} \in \Omega. \tag{4.59}$$

Similarly, the traction tensor $\boldsymbol{\tau} = \boldsymbol{T} \cdot \boldsymbol{n}$ used by the boundary conditions is implemented with the following stress tensor:

$$\boldsymbol{T} = \eta[\nabla\boldsymbol{v} + (\nabla\boldsymbol{v})^T] + \eta(\frac{2\nu}{1-2\nu}\nabla \cdot \boldsymbol{v})\boldsymbol{I}. \tag{4.60}$$

In both equations, the Poisson's ratio ν controls the incompressibility. When $\nu \to 0.5$, these two equations converge back to Eqns. (4.44, 4.45, 4.47). Note that λ in the linear elasticity equations has been replaced with $\frac{2\eta\nu}{1-2\nu}$ from the relation $\nu = \frac{\lambda}{2(\eta+\lambda)}$.

While we choose to model quasi-incompressible Stokes with an analogy between linear elasticity and Stokes, it is worth pointing out that using the saddle point formulation for discretizing the truly incompressible Stokes flow is still a viable technique. In fact, it would be recommended when paired with an iterative sparse linear solver like Preconditioned Conjugate Gradient (PCG) or multigrid methods. We stress that we opt to use the quasi-incompressible formulation due to our reliance on direct sparse solvers, whose advantages over iterative solvers will become evident in gradient computation (Sec. 4.2.6). Furthermore, there is a much higher degree of comfort and experience in standard topology optimization with "stock" linear elasticity, while Stokes systems are not as widespread.

4.2.4 Discretization

We discretize our governing equations on a Cartesian background grid that embeds the geometry of the fluid cavity as in Fig. 4-21 (a) (as opposed to using a mesh that conforms to the boundary of the fluid container). We employ a collocated discretization where all components of the velocity field are stored at the same locations, at the nodes of the Cartesian grid, and since we use Eqn. (4.59) for our quasi-incompressible

Stokes fluid, there is no need to involve any "pressure" state variables in our formulation. Using a variational approach, we can express boundary conditions at a sub-grid resolution while only storing variables at regular grid-node locations. Again, we stress that due to the strong resemblance between Eqn. (4.59) and its linear elasticity counterpart Eqn. (4.54), the numerical discretization paradigm to be discussed in this section is essentially the commonly used discretization scheme in linear elasticity, allowing practitioners to reuse implementations in topology optimization with little extra effort.

Variational form and embedded traction boundaries We initially focus on the quasi-incompressible Stokes problem in a domain $\Omega \in \mathcal{R}^d$, under *traction* boundary conditions, stated as follows:

$$-\nabla \cdot \boldsymbol{T} = -\eta \boldsymbol{\Delta} \boldsymbol{v}(\boldsymbol{x}) - \frac{\eta}{1-2\nu}\nabla[\nabla \cdot \boldsymbol{v}(\boldsymbol{x})] = \boldsymbol{f}(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Omega, \tag{4.61}$$

$$\boldsymbol{T} \cdot \boldsymbol{n} = \boldsymbol{\beta}(\boldsymbol{x}) \qquad \boldsymbol{x} \in \partial\Omega, \tag{4.62}$$

where the stress tensor $\boldsymbol{T}$ is defined as in Eqn. (4.60). It is known [71, 34] that the associated variational formulation of this problem computes the solution via minimization of an energy functional $E[\boldsymbol{v}]$ over all functions $\boldsymbol{v}$ in an appropriate solution space. For our purposes, we define the solution space to be all functions defined by bilinear (2D) or trilinear (3D) interpolation over the cells of the background Cartesian grid, and the associated energy to be minimized [34, 168] is

$$E[\boldsymbol{v}] = \int_\Omega \Psi[\boldsymbol{v}(\boldsymbol{x})]d\boldsymbol{x} - \int_\Omega (\boldsymbol{v} \cdot \boldsymbol{f})d\boldsymbol{x} - \int_{\partial\Omega} (\boldsymbol{v} \cdot \boldsymbol{\beta})dS, \tag{4.63}$$

where the energy density $\Psi[\boldsymbol{v}]$ is defined as follows:

$$\Psi[\boldsymbol{v}] = \eta\|\boldsymbol{D}[\boldsymbol{v}]\|_F^2 + \frac{\eta\nu}{1-2\nu}[\mathrm{tr}(\boldsymbol{D}[\boldsymbol{v}])]^2,$$

with the strain rate $\boldsymbol{D}$ computed from $\boldsymbol{v}$ as in Eqn. (4.46). We note that in most of our examples we do not use any external forces (e.g. gravity), hence $\boldsymbol{f} = 0$, and only use zero-valued (or open-boundary) traction boundary conditions, thus $\boldsymbol{\beta} = 0$. As a consequence, the last two integrals in Eqn. (4.63) are zero for our examples. Should it be necessary, however, to incorporate non-zero forces or traction conditions in a different application, these terms can trivially be included in our discretization, and we later discuss how both volume and boundary integrals can be computed via quadrature within our solution space.

Sub-cell energy quadrature Using bilinear/trilinear interpolation as shown in Fig. 4-21 (b), our solution space contains all functions of the form

$$\hat{\boldsymbol{v}}(\boldsymbol{x}; \mathcal{V}) = \sum_i \boldsymbol{v}_i \mathcal{N}_i(\boldsymbol{x}), \tag{4.64}$$

where $\mathcal{N}_i(\boldsymbol{x})$ is the shape function associated with grid node i, and $\mathcal{V} = \{\boldsymbol{v}_i\}$ collectively represents all nodal velocities in our grid. Using this interpolation, we can also express all derivative quantities as functions of the nodal velocities, by proper manipulation of the shape functions. For example, the entries of the strain rate tensor $\hat{\boldsymbol{D}}(\boldsymbol{x};\mathcal{V}) = \boldsymbol{D}[\hat{\boldsymbol{v}}(\boldsymbol{x};\mathcal{V})]$ are evaluated as

$$\hat{\boldsymbol{D}}_{pq} = \frac{1}{2}\sum_i \left[\boldsymbol{v}_i^{(p)}\mathcal{N}_{i,q}(\boldsymbol{x}) + \boldsymbol{v}_i^{(q)}\mathcal{N}_{i,p}(\boldsymbol{x})\right], \tag{4.65}$$

where superscripts in parentheses for $\boldsymbol{v}$ indicate coordinate components, and subscripts in shape functions after commas indicate partial derivatives. We can continue this substitution to express the energy density and ultimately the integrated energy in Eqn. (4.63) as a function of the nodal velocity values. Since $\hat{\boldsymbol{D}}$ is a linear function of nodal velocities, and the energy density Ψ has a quadratic dependence on $\boldsymbol{D}$, the overall energy $E[\mathcal{V}] = E[\hat{\boldsymbol{v}}]$ will ultimately reduce to a quadratic convex function over the nodal velocities, with the coefficients of this polynomial involving integrals of products of derivatives of the shape functions. All of this is, of course, merely a restatement of the standard finite element discretization approach in a Cartesian lattice [71, 115].

The integral in Eqn. (4.63) can be computed on a per-cell basis; using our assumption that $\boldsymbol{f} = \boldsymbol{\beta} = \boldsymbol{0}$, we can write

$$E[\mathcal{V}] = \int_\Omega \Psi[\hat{\boldsymbol{v}}(\boldsymbol{x};\mathcal{V})]d\boldsymbol{x} = \sum_k \underbrace{\int_{\Omega\cap\mathcal{C}_k} \Psi[\hat{\boldsymbol{v}}(\boldsymbol{x};\mathcal{V})]d\boldsymbol{x}}_{E_k[\mathcal{V}]}, \tag{4.66}$$

where the summation is taken over all cells $\{\mathcal{C}_k\}$ in our background grid. The per-cell energies E_k fall in one of two categories. For fully interior cells (for which $\mathcal{C}_k \subset \Omega$) the integral can be computed exactly either via analytic integration (the integrands are low-degree polynomials), or with a 4-point (8-point in 3D) Gauss quadrature; this yields the same stencil that is used in several similar methods [14, 1, 95]. For boundary cells (those that have $\mathcal{C}_k \cap \partial\Omega \neq \emptyset$) we must specify a quadrature rule for the partial-cell domain of integration $\mathcal{C}_k \cap \Omega$.

We propose a quadrature rule for such boundary cells motivated by the following design objectives: (a) We seek a rule that is as simple as possible, so as to be easily adaptable to scenarios where the boundary location is evolving, as in the context of shape optimization, (b) The rule must give rise to continuous solutions as the boundary evolves, to ensure differentiability of such solution with respect to design parameters, and (c) The rule should have at least a rudimentary degree of accuracy (e.g. exactly integrate constant integrands) and be free of common defects. In light of these design traits, we propose a quadrature formula that uses four weighted quadrature points (as shown in Fig. 4-21 (c) in 2D), placed at the centers of four equal quadrants $\mathcal{C}_k^{(0)}, \ldots, \mathcal{C}_k^{(3)}$ produced by bisecting the boundary cell along each axis (see Fig. 4-21 (c)). If we denote by $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_3$ the centers of these quadrants and by $\Omega_0 := \mathcal{C}_k^{(0)} \cap \Omega, \ldots, \Omega_3 := \mathcal{C}_k^{(3)} \cap \Omega$ the fractions of these quadrants that are interior to the fluid domain Ω, the quadrature rule becomes

$$\int_{\Omega\cap\mathcal{C}_k}\Psi[\hat{\boldsymbol{v}}(\boldsymbol{x};\mathcal{V})]d\boldsymbol{x}\approx\sum_{j=0}^{3}\mathrm{Area}(\Omega_j)\Psi[\hat{\boldsymbol{v}}(\boldsymbol{x}_j;\mathcal{V})]. \tag{4.67}$$

A similar quadrature rule would naturally be defined in 3D, using eight quadrature points at the center of the octants that a cell is split by bisecting each axis, weighted by the corresponding volume fraction of each that falls inside Ω. It is clear that the quadrature rule integrates constant functions exactly (due to the area factors), and that it would provide for continuously varying solutions as the boundary evolves (as the minimizers of a convex quadratic with continuously varying coefficients). The use of multiple quadrature points (as opposed to a single one, at the centroid of the cell $\mathcal{C}_k$) is mandated in order to avoid hourglassing instabilities in the discretization [104]. However, keeping this quadrature rule simple by only having the area factors dependent on the geometry of Ω greatly simplifies the task of differentiating our flow solution with respect to the design parameters, as we see in Sec. 4.2.6. We have found this formulation to be effective and sufficient in our examples, and as we see next it can be used in conjunction with the other boundary condition types we need in our application.

Dirichlet boundary conditions The formulation of the preceding paragraph is sufficient to accommodate pure *traction* boundary conditions, as the open boundary conditions at the outlet of the fluidic device in Fig. 4-21 (a) in blue. In addition, we can easily accommodate Dirichlet boundary conditions imposed precisely at grid nodes, by simply setting them to a constant value while minimizing $E[\boldsymbol{v}]$. A more challenging, but essential scenario to accommodate would be the enforcement of Dirichlet conditions on an *embedded boundary* rather than one that is aligned with the grid faces. Such an example would correspond to the lateral edges of the device in Fig. 4-21 (a) in brown, should a no-slip Dirichlet condition ($\boldsymbol{v} = \mathbf{0}$) have been imposed.

Enforcement of such embedded Dirichlet conditions is not quite straightforward with variational formulations, as opposed to traction conditions (analogous to *Neumann* conditions in the Poisson's equation) which are naturally incorporated into the energy $E[\boldsymbol{v}]$. Possible options such as a "soft" constraint enforcement [132, 86] have to contend with ad-hoc constraint stiffnesses, while imposing Dirichlet conditions at zero crossings between the interface and grid edges is known to be questionable in its convergence quality or even the existence of a compliant solution [108, 12]. Instead, we employ a formulation for the enforcement of embedded Dirichlet conditions that leverages a weak formulation of the constraint using an appropriate approximation space [64, 168], that has been successfully used with elliptic PDEs in similar contexts.

Let $\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{\alpha}(\boldsymbol{x})$ be the Dirichlet condition we want enforced in a section of the boundary $\Gamma_D \subset \partial\Omega$ that is intersecting grid cells, rather than being aligned with edge boundaries. Following previous work [12, 168], for each such cell, we enforce the

Dirichlet condition in an *averaged* fashion, via the integral constraint

$$\int_{\mathcal{C}_k \cap \Gamma_D} \hat{\boldsymbol{v}}(\boldsymbol{x}; \mathcal{V}) d\boldsymbol{x} = \int_{\mathcal{C}_k \cap \Gamma_D} \boldsymbol{\alpha}(\boldsymbol{x}) d\boldsymbol{x}$$

which, given the expansion of $\hat{\boldsymbol{v}}$ using the shape functions, becomes

$$\sum_i \boldsymbol{v}_i \int_{\mathcal{C}_k \cap \Gamma_D} \mathcal{N}_i(\boldsymbol{x}) d\boldsymbol{x} = \int_{\mathcal{C}_k \cap \Gamma_D} \boldsymbol{\alpha}(\boldsymbol{x}) d\boldsymbol{x}. \tag{4.68}$$

Eqn. (4.68) is effectively a (d-dimensional) linear equality constraint associated with each boundary cell. The integral of the shape function on the left-hand side is computed analytically via a hyperplane approximation to the cell boundary $\mathcal{C}_k \cap \Gamma_D$. We construct a best-fit line in 2D (plane in 3D) to this boundary section based on the signed distances from grid nodes to the boundary. We use a quadrature rule to approximate the integral of both the shape function and $\boldsymbol{\alpha}$ over $\mathcal{C}_k \cap \Gamma_D$ unless the integral is trivial to compute analytically, e.g., $\boldsymbol{\alpha}(\boldsymbol{x})$ is constant. In our application, only a no-slip, zero-Dirichlet boundary condition $\boldsymbol{\alpha}(\boldsymbol{x}) = \boldsymbol{0}$ is employed, hence the integral on the right-hand side is trivially zero. We discuss this quadrature rule in greater detail in Sec. 4.2.5 and our supplemental material.

Aggregating all such constraints from all cells that intersect the Dirichlet boundary yields a linear system of constraints $\boldsymbol{C}\boldsymbol{v} = \boldsymbol{d}$ (for simplicity of notation we use here the symbol $\boldsymbol{v}$ for *all* nodal velocities, in replacement of $\mathcal{V}$). As mentioned before, for no-slip boundaries we would have $\boldsymbol{d} = \boldsymbol{0}$.

No-separation, zero-friction boundaries Although no-slip conditions can be accommodated as in the previous paragraph, enforcing no-separation boundary conditions combined with a zero tangential component of the traction vector is the norm in our examples, as encoded in Eqns. (4.50, 4.51). Similar to the previous paragraph, the projected Dirichlet condition $\boldsymbol{v} \cdot \boldsymbol{n} = 0$ is enforced via an integral constraint

$$\int_{\mathcal{C}_k \cap \Gamma_D} \hat{\boldsymbol{v}}(\boldsymbol{x}; \mathcal{V}) \cdot \boldsymbol{n} d\boldsymbol{x} = \sum_i (\boldsymbol{v}_i \cdot \boldsymbol{n}) \int_{\mathcal{C}_k \cap \Gamma_D} \mathcal{N}_i(\boldsymbol{x}) d\boldsymbol{x} = 0 \tag{4.69}$$

which is now just a single scalar constraint (per cell) as shown in Fig. 4-21 (d), while the zero tangential component of the traction is implicitly enforced from the energy formulation in Eqn. (4.63). Again, a single constraint system of the form $\boldsymbol{C}\boldsymbol{v} = \boldsymbol{d}$ can aggregate all boundary conditions other than traction boundaries (which are incorporated in the energy), including (a) node-aligned Dirichlet constraints, (b) embedded Dirichlet constraints, and (c) no-separation conditions.

4.2.5 Forward Simulation

Given the discretization described in Sec. 4.2.4, we now provide a means of computing the fluid velocity field $\boldsymbol{v}$ given the boundary shape parameterized by a vector of parameters $\boldsymbol{\theta}$. The specific definition of $\boldsymbol{\theta}$ depends on the type of parametric surfaces. Additionally, if multiple parametric surfaces exist in the problem domain, their parameters are aggregated into a single vector $\boldsymbol{\theta}$. As described in Sec. 4.2.4, $\boldsymbol{v}$ is the minimizer of the variational form of the energy in Eqn. (4.63) with boundary conditions applied. Due to the analogy between Stokes flow and linear elasticity, the post-discretization variational form of the energy, known to be quadratic in $\boldsymbol{v}$, can be defined as $\frac{1}{2}\boldsymbol{v}^T\boldsymbol{K}\boldsymbol{v}$ where $\boldsymbol{K}$ has an equivalent role of the stiffness matrix in linear elasticity. Further, the discretized boundary conditions, already introduced as $\boldsymbol{C}\boldsymbol{v} = \boldsymbol{d}$, are linear in $\boldsymbol{v}$. Combined, these form a convex quadratic programming (QP) problem, the solution to which describes the fluid velocity

$$\tilde{\boldsymbol{v}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{v}} \quad \frac{1}{2}\boldsymbol{v}^T\boldsymbol{K}(\boldsymbol{\theta})\boldsymbol{v}. \tag{4.70}$$

$$\text{s.t.} \quad \boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{v} = \boldsymbol{d}(\boldsymbol{\theta}). \tag{4.71}$$

Here, the convexity comes from the positive semi-definiteness of the stiffness matrix $\boldsymbol{K}$. The notation $\tilde{\boldsymbol{v}}$ is used to emphasize that this is the minimizer of the QP problem, dependent on the parameter $\boldsymbol{\theta}$. Meanwhile, $\boldsymbol{K}, \boldsymbol{C}$, and $\boldsymbol{d}$ are written as functions of $\boldsymbol{\theta}$ as their values are dependent on the location of the solid-fluid interface boundaries. This should be especially obvious in the case of $\boldsymbol{K}$, as in Eqn. (4.67) we can see that boundary cells contribute to this term by an amount proportional to the area of their region of overlap with Ω. Again, we point out that the external force $\boldsymbol{f}$ and traction condition $\boldsymbol{\beta}$ are ignored for simplicity. Should it be necessary, both of them could be easily added back to Eqn. (4.70) as a linear term on $\boldsymbol{v}$ with its linear weights dependent on $\boldsymbol{\theta}$.

The remainder of this section is dedicated to describing how $\boldsymbol{K}$, $\boldsymbol{C}$, and $\boldsymbol{d}$ are computed from $\boldsymbol{\theta}$, concretely describing how to calculate the steady-state flow and laying the groundwork for the gradient computation. Given a set of design parameters $\boldsymbol{\theta}$, the analytic signed distance function of the boundary is computed at each cell corner, building a signed-distance field on the whole grid. We then compute a hyperplane of best fit (line in 2D; plane in 3D) to approximate the geometry of the boundary within each individual cell (where applicable). This hyperplane is used to compute the stiffness matrix component $\boldsymbol{K}^{ij}$ for the cell with indexing (i, j). Further, we use it to integrate the shape function $\mathcal{N}_i$ and the boundary condition $\boldsymbol{\alpha}$ described in Sec. 4.2.4 to form the linear constraints $\boldsymbol{C}\boldsymbol{v} = \boldsymbol{d}$. With the full QP formulated, $\tilde{\boldsymbol{v}}$ is obtained by solving the KKT conditions.

Signed-distance functions (SDFs) for parametric shapes Given design parameters $\boldsymbol{\theta}$ that determine the fluid-solid boundary, we first compute the signed distance from each cell's corners to the boundary. With the type of parametric surface known beforehand, evaluating this distance typically involves nothing more than analyzing the functions describing the level-set of the parametric shape. For example, if $\boldsymbol{\theta}$ parameterizes a circle with $\boldsymbol{\theta} = (\boldsymbol{c}, r)$ (the center position and radius of the circle), then the signed distance function can be written compactly as $\phi(\boldsymbol{x}) = r - \|\boldsymbol{c} - \boldsymbol{x}\|_2$ for any $\boldsymbol{x}$. Signed-distance functions of more sophisticated parametric shapes, *e.g.*, Bézier curves, can be found in our supplemental material. Throughout this paper, we use the convention that $\phi(\boldsymbol{x}) > 0$ refers to the solid region and $\phi(\boldsymbol{x}) < 0$ corresponds to the fluid region.

Boundary in a cell Once the signed distances from a cell's corners to the boundary are given, the next step is to fit a hyperplane (line in 2D and plane in 3D) that approximates the fluid-solid interface in the cell when necessary. Note that this implicitly assumes the boundary does not contain delicate structures that are significantly smaller than the cell size. Depending on the signs at each corner, a cell is classified into three categories: purely in the interior of the solid region, purely in the interior of the fluid region, or partially in both regions. Only this final case requires fitting a hyperplane inside the cell, for which we obtain the hyperplane parameters from a linear least squares regression on the signed distances at its corners. As techniques for solving linear least squares are mature, we leave the details in our supplemental material.

At the end of this step, we have determined the type of each cell, and, for mixed-cells, we have provided an analytic means of approximating the boundary with a hyperplane. Such hyperplanes are crucial in assembling the matrices and vectors in Eqns. (4.70, 4.71).

Assembling $\boldsymbol{K}$ Computing $\boldsymbol{K}$ requires reasoning about two different types of cells: fluid cells and mixed cells. In the former case, the procedure for computing the contribution to $\boldsymbol{K}$ is well-established in the linear elasticity literature [14], as the integrals involved are evaluated over entire cells, either analytically or via Gauss quadrature rules. In the latter, mixed-cell case, it can be seen from Eqn. (4.67) that dependence on $\boldsymbol{\theta}$ only occurs via the area term. This is because the energy density function is evaluated at the same quadrature locations regardless of the cell type. The area function describes the ratio of the cell that is fluid and can be computed compactly with a single, closed-form expression [9] using the hyperplanes computed before:

Proposition 1. *Consider a d-dimensional halfspace $H = \{\boldsymbol{x} | \boldsymbol{a} \cdot \boldsymbol{x} + b \geq 0\}$ with the assumption that $\Pi_i a_i \neq 0$. The volume of its intersection with a unit hypercube is*

$$|[0,1]^d \cap H| = \sum_{\boldsymbol{q} \in F^0 \cap H} \frac{(-1)^{|\boldsymbol{q}_0|}(\boldsymbol{a} \cdot \boldsymbol{q} + b)^d}{d! \Pi_i a_i} \tag{4.72}$$

where $F^0 = \{0,1\}^d$ is the set of all hypercube corners and $|\boldsymbol{q}_0|$ is the number of zeros in the entries of $\boldsymbol{q}$.

Since this solution is closed-form and analytical, gradients can be simply and easily computed, which is especially beneficial in 3D, where plane-cube intersections can lead to complex cell fluid geometries.

Assembling $\boldsymbol{C}$ and $\boldsymbol{d}$ The remaining step is to compute $\boldsymbol{C}$ and $\boldsymbol{d}$ as a function of $\boldsymbol{\theta}$. We do this on a row-wise basis, again focusing on the (nontrivial) mixed cells. As established in Sec. 4.2.4, computing $\boldsymbol{C}$ and $\boldsymbol{d}$ requires computing the integral of the shape function $\mathcal{N}_i$ over the cross-sectional area of the boundary and the cell. Computing this integral analytically is tedious particularly in 3D because the cross-sectional area can have anywhere from three to six edges, and the situation would become even worse when computing the gradients. Thus (and keeping our procedure general), we design the following quadrature rule to approximate this line or area integral. Beginning with the quadrature points $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{2^d-1}$ which are in the centers of the cell quadrants (octants in 3D), we first project $\boldsymbol{x}_i$ onto the fluid-solid boundary approximated by our hyperplanes. In order to integrate a function over the boundary in the cell, we use these projections as the quadrature points to approximate the integral in Eqn. (4.68):

$$\int_{\mathcal{C}_k \cap \Gamma_D} \mathcal{N}_i(\boldsymbol{x}) d\boldsymbol{x} \approx \sum_{j=0}^{2^d-1} \operatorname{Proj}(\boldsymbol{x}_j; \boldsymbol{\theta}) \operatorname{Area}(\mathcal{C}_k^{(j)} \cap \partial\Omega), \tag{4.73}$$

where $\operatorname{Proj}(\cdot; \boldsymbol{\theta})$ is an operator that projects a point onto the hyperplane and its reliance on $\boldsymbol{\theta}$ is due to the hyperplane parameters. The area function evaluates the cross-sectional area of the fluid-solid boundary and the quadrants or octants $\mathcal{C}_k^{(j)}$. We calculate this area factor by noticing it is the directional derivative of $\operatorname{Area}(\mathcal{C}_k^{(j)} \cap \Omega)$ along the hyperplane's normal:

$$\operatorname{Area}(\mathcal{C}_k^{(j)} \cap \partial\Omega) = \frac{\partial \operatorname{Area}(\mathcal{C}_k^{(j)} \cap \Omega)}{\partial b} \|\boldsymbol{a}\|_2, \tag{4.74}$$

where $\boldsymbol{a}$ and b are defined as in Prop. 1. Computing this directional derivative requires nothing more than directly applying the chain rule to Prop 1, and we leave its details in our supplemental material.

Solving the QP problem Having assembled every piece of Eqns. (4.70, 4.71), we demonstrate how to compute $\tilde{\boldsymbol{v}}$ as a function of $\boldsymbol{\theta}$. Since $\boldsymbol{K}$ (analogous to an elastic stiffness matrix) is positive semi-definite, the quadratic term is guaranteed to be convex. Thus, a (global) minimum for $\tilde{\boldsymbol{v}}$ always exists. We solve this QP by solving the KKT system:

$$\begin{pmatrix} \boldsymbol{K}(\boldsymbol{\theta}) & \boldsymbol{C}^T(\boldsymbol{\theta}) \\ \boldsymbol{C}(\boldsymbol{\theta}) & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{v}} \\ \tilde{\boldsymbol{\lambda}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{d}(\boldsymbol{\theta}) \end{pmatrix}, \tag{4.75}$$

where $\tilde{\boldsymbol{\lambda}}$ is a Lagrange multiplier. We choose to solve this KKT system with a direct factorization method rather than apply an iterative optimization algorithm, as the pre-factorization of this system can help accelerate the gradient computation in the next section. Thus, solving $\tilde{\boldsymbol{v}}$ reduces to solving a symmetric (possibly indefinite) linear system depending purely on $\boldsymbol{\theta}$.

Table 4.5: A summary of our design problems. The "Time (s)/Call" column reports the average wall-clock time of one function call to compute forward simulation and backpropagation. The time was measured on a single Intel(R) Xeon(R) CPU E7-4830 v4 @ 2.00GHz. The "Final loss" column reports the loss of our optimized design. The loss functions in all problems are normalized such that a unit loss refers to the average performance of randomly sampled designs and zero loss means an oracle design that perfectly matches the desired target, which may not exist in some problems.

Name	Grid resolution	# Param.	Level set
Amplifier	64×48	5	Béizer curves
Flow Averager	$64 \times 64 \times 4$	8	Béizer curves
Funnel	$64 \times 64 \times 16$	10	Béizer curves
Superposition Gate	$64 \times 64 \times 4$	5	Béizer curves
Fluidic Twister	$64 \times 64 \times 32$	32	NURBS surface
Fluidic Switch	$64 \times 64 \times 32$	26	NURBS surface

Name	Boundary condition	Time (s)/Call	Final loss
Amplifier	No-separation	0.2	1.7e-5
Flow Averager	No-separation	5.3	3.1e-2
Funnel	No-separation	64.8	2.3e-1
Superposition Gate	No-separation	4.9	4.9e-1
Fluidic Twister	No-separation/No-slip	56.1	4.9e-2/9.7e-1
Fluidic Switch	No-slip	171.1	5.8e-1

4.2.6 Optimization

Given design parameters $\boldsymbol{\theta}$, we have described how to perform forward simulation in order to compute the steady-state velocity field $\tilde{\boldsymbol{v}}$. We now detail how to compute the backward gradient computation, i.e. computing the derivative of the loss function with respect to $\boldsymbol{\theta}$. We begin this section by defining loss functions evaluating the flow generated by simulation. Next, we discuss how gradients are computed via a scheme that back-propagates through the simulation. We conclude with a description of the full optimization algorithm.

Loss functions While our method imposes no restrictions on the loss function as long as its gradients are well defined, we focus on a specific family of loss functions that penalize the discrepancy between the desired and actual velocity fields $\tilde{\boldsymbol{v}}$:

$$L(\tilde{\boldsymbol{v}}) = \|F(\tilde{\boldsymbol{v}}) - F(\boldsymbol{v}^*)\|_p, \tag{4.76}$$

where $\boldsymbol{v}^*$ is a target velocity field, F is a function that extracts features we are interested in optimizing from a velocity field, and $\|\cdot\|_p$ is the p-norm. The choice of F is flexible and problem-specific. For example, F can be a selector function that picks velocities at the outlet of the device only, or F can be a curl operator for tasks focusing on optimizing the rotational speed of a velocity field.

Gradient computation Given a complete description of the forward simulation scheme, deriving the gradients – at a high level – requires no more than iterative application of the chain rule. Each step in forward simulation has a corresponding step in the gradient computation that is then chained together. Most of these steps require the straightforward computation of the gradient of the output of the forward simulation (of that step) with respect to the input (of that step). Therefore, we leave the details of gradient computation in our supplemental material and only highlight one key step: the gradients of the solution of the QP problem. Specifically, we describe the computation of $\partial\tilde{\boldsymbol{v}}/\partial\boldsymbol{K}$, $\partial\tilde{\boldsymbol{v}}/\partial\boldsymbol{C}$, and $\partial\tilde{\boldsymbol{v}}/\partial\boldsymbol{d}$. In order to avoid unwieldy, high-dimensional tensor notation, we describe the gradient derivation in differential form (sufficient for the purpose of computing the gradients). Concretely, given perturbations $\delta\boldsymbol{K}$, $\delta\boldsymbol{C}$, and $\delta\boldsymbol{d}$, we explain how much perturbation $\delta\tilde{\boldsymbol{v}}$ is expected.

Differentiation the KKT system in Eqn. (4.75) results in the following linear system with $\delta\tilde{\boldsymbol{v}}$ and $\delta\tilde{\boldsymbol{\lambda}}$ as unknowns (we omit the $\boldsymbol{\theta}$ dependence for clarity):

$$\begin{pmatrix}\boldsymbol{K} & \boldsymbol{C}^T\\ \boldsymbol{C} & \boldsymbol{0}\end{pmatrix}\begin{pmatrix}\delta\tilde{\boldsymbol{v}}\\ \delta\tilde{\boldsymbol{\lambda}}\end{pmatrix}=\begin{pmatrix}\boldsymbol{0}\\ \delta\boldsymbol{d}\end{pmatrix}-\begin{pmatrix}\delta\boldsymbol{K} & \delta\boldsymbol{C}^T\\ \delta\boldsymbol{C} & \boldsymbol{0}\end{pmatrix}\begin{pmatrix}\tilde{\boldsymbol{v}}\\ \tilde{\boldsymbol{\lambda}}\end{pmatrix}. \tag{4.77}$$

$\delta\boldsymbol{K}$, $\delta\boldsymbol{C}$, and $\delta\boldsymbol{d}$ are all known; $\tilde{\boldsymbol{v}}$ and $\tilde{\boldsymbol{\lambda}}$ have been computed during the forward simulation process. Since the linear system of equations here has precisely the same left-hand side as the KKT system solved in the forward simulation, this matrix can be pre-factorized once during simulation and reused during gradient computation, allowing for efficient solving of $\delta\tilde{\boldsymbol{v}}$.

Optimization With a method for computing gradients of the loss backward through simulation with respect to design parameters in tow, we are able to apply gradient-based, quasi-Newton methods for efficient optimization. The performance of this approach primarily depends on two crucial factors: the specific local optimization method chosen, and the initial guess. Since all of our design problems have nonlinear continuous losses and bound constraints on parameters only, we chose L-BFGS, a classic quasi-Newton method, as our optimizer. The initial guess was selected by picking the best design among a number of randomly sampled designs in the design space. Sampling designs serves two purposes in our method: first, it reduces the risk of getting trapped into local minima. Second, we can rescale the loss function in each design problem by setting the average loss from these randomly samples as the unit loss. After this normalization, the loss functions from different design problems share the same physical meaning: loss = 0 means an oracle solution that matches the target perfectly, which is not always possible, and loss = 1 means the solution has an empirically average performance among all possible designs.

4.2.7 Results and Discussions

In this section, we present six 2D and 3D design problems to evaluate the performance of our implementation of the differentiable Stokes flow as well as the optimization pipeline. We start this section by describing the problem statements for each design problem, followed by evaluations and discussions on the experimental results. We ask readers to refer to our supplemental video for a complete demonstration of these design problems, the evolution of our optimization process, and the animation of our final results.

Design Problems

We summarize the basic information about these design problems in Table 4.5 and Fig. 4-25, 4-22, and 4-23. The number of decision variables in these design problems ranges from 5 to 32, and the cell resolution of the scenes varies from 64×48 cells in 2D to $64 \times 64 \times 32$ cells in 3D. Below we discuss the setup of each design problem in detail:

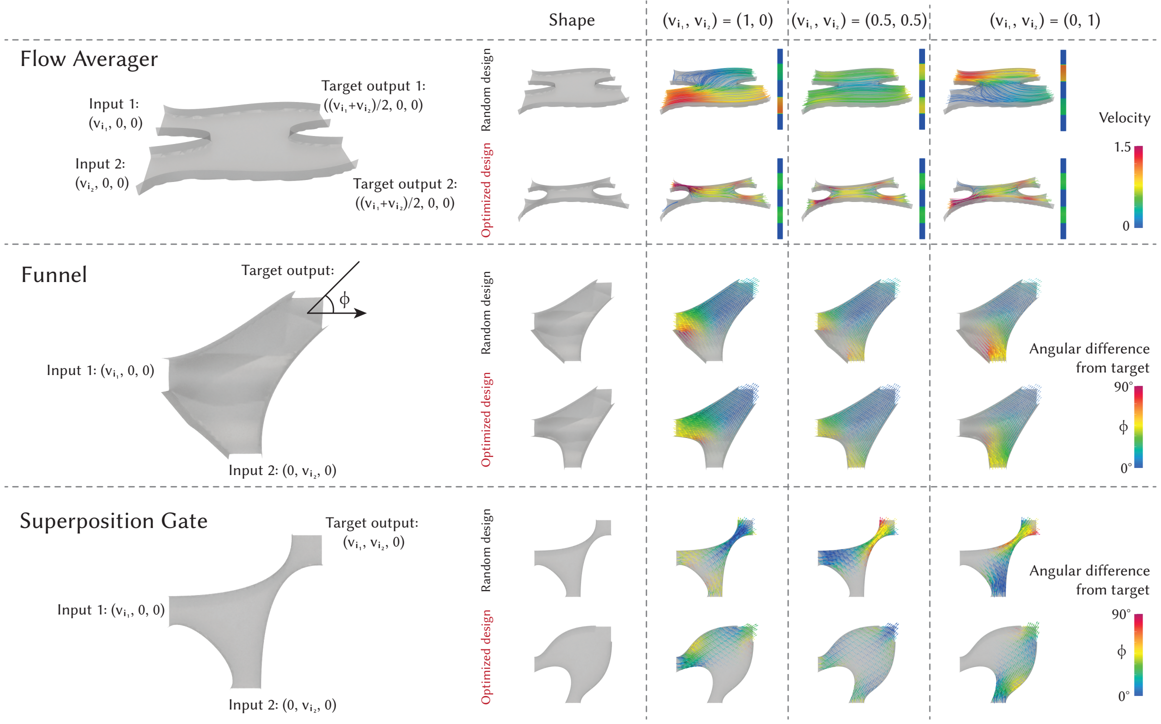


Figure 4-22: Three optimization examples: the Flow Averager, the Funnel, and the Superposition Gate. The left figure of each example shows the specifications of the design problem. The right eight figures of each example show the comparison between a randomly sampled design (top row) and the optimized design (bottom row) with three different inputs. In the Flow Averager, the vertical color bar inside each inset indicates the velocity magnitude at the cross section of the two outlets, and solutions with two outlets having more similar colors are better. In the Funnel and the Superposition Gate, the color indicates the angular error between the local velocity and the target velocity (cooler at the outlet is better).

Amplifier This motivating example in Fig. 4-20 is a 2D design problem that aims to amplify the velocity of inlet flow by a factor of 3. The design variables are the control points of the Bézier curves representing the upper and lower solid-fluid boundaries.

The inlet flow enters the domain from the left with a velocity of $(v, 0)$, and the loss function is defined as the difference between $(3v, 0)$ and the average speed of the outlet flow on the right.

Flow Averager The goal of this design problem is to engineer a fluidic load balancer with two inputs (left) and two outputs (right). Let the two inlet flows enter the domain with velocities $(v_{i_1}, 0, 0)$ and $(v_{i_2}, 0, 0)$ where v_{i_1} and v_{i_2} are arbitrary numbers and let $\boldsymbol{v}_{o_1}$ and $\boldsymbol{v}_{o_2}$ be the average flow velocities at the two outlets. The objective is to encourage both $\boldsymbol{v}_{o_1}$ and $\boldsymbol{v}_{o_2}$ to be as close as possible to $((v_{i_1} + v_{i_2})/2, 0, 0)$ (Fig. 4-22 top). In other words, we expect the design to average the two inputs no matter what values are given for v_{i_1} and v_{i_2}. We optimize a loss function that concurrently optimizes for these two basis inputs $(v_{i_1}, v_{i_2}) = (0, 1)$ and $(1, 0)$. The design space consists of four Bézier curves in 2D. The 3D solid-fluid boundaries are formed by extruding these curves vertically.

Funnel This design problem considers a fluidic domain with two inputs and one output. The goal is to design the internal boundary of the fluidic domain such that the direction of the output flow is 45-degrees and input-invariant (Fig. 4-22 middle). Let $(v_{i_1}, 0, 0)$ and $(0, v_{i_2}, 0)$ be the two inlet flow velocities; the design is evaluated by continuously varying the inputs from $(v_{i_1}, v_{i_2}) = (1, 0)$ and $(0, 1)$ and observing the change in the direction of the outlet flow. Note that while the design space consists of 2D Béizer curves only, the bumpy bottom plate creates enough vertical variations to make it our first 3D design problem.

Superposition Gate This design problem shares the similar setting with the Funnel above except that the goal is to obtain an outlet flow with a velocity of $(v_{i_1}, v_{i_2}, 0)$ (Fig. 4-22 bottom); thus the name superposition gate. When the inputs (v_{i_1}, v_{i_2}) vary from $(1, 0)$ to $(0, 1)$, an ideal superposition gate design should generate an outlet flow that sweeps the first quadrant.

Fluidic Twister This 3D problem considers designing the internal surface of a tunnel to generate a twisted flow (Fig. 4-23). With a straight inlet flow $\boldsymbol{v}_i = (0, 0, -1)$ entering the tunnel from the top, an ideal design needs to generate an outlet velocity field $\boldsymbol{v}_o = (u_o, v_o, w_o)$ at the bottom such that it has a desired vertical curl ω: $\nabla \times (u_o, v_o, 0) = (0, 0, 2\omega)$. We discretize the curl operator on our grid and define the loss function as the difference between $\nabla \times (u_o, v_o, 0)$ and $(0, 0, 2\omega)$. The internal surface of the tunnel is represented by a NURBS surface with 32 free control points to be optimized.

Fluidic Switch In this problem, we consider a fluidic device with a switch that can kinematically change the fluid-solid boundary in a fluidic domain. By switching on and off, we expect the outlet flow velocity from the device to transition between two prescribed velocity profiles (Fig. 4-25). We set up our fluidic domain with one input, two outputs, and a solid obstacle immersed in the fluidic domain. The solid obstacle is

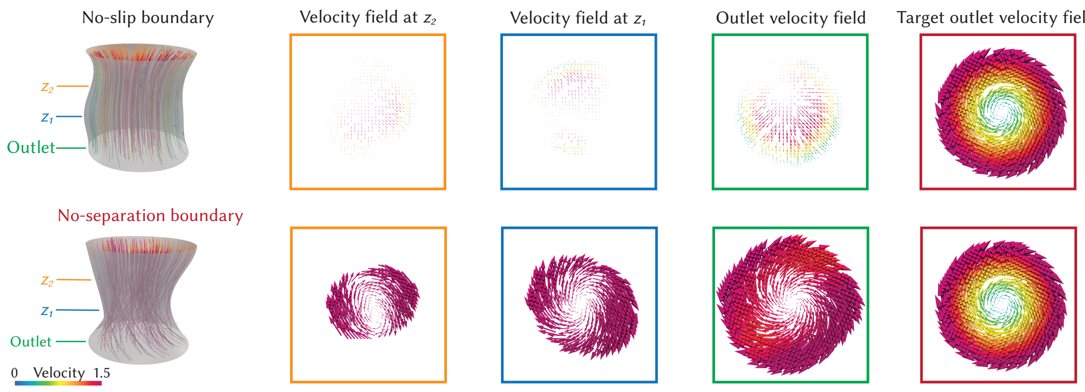


Figure 4-23: A comparison between optimizing the Fluidic Twister with no-slip boundaries (top) and no-separation boundaries (bottom). We show the velocity field at three cross-sectional areas (middle three columns) of the optimized design (left column) as well as the target, twisted field (right column) at the outlet (green). With the no-slip boundary, the resulting velocity field is attenuated significantly. With the no-separation boundary, a desired helical pattern emerges to facilitate the swirling of the outlet flow.

parameterized by a NURBS surface whose 24 control points are to be optimized, and it is pinned on the bottom plate so it can rotate along a vertical axis. The two states of the switch set the rotational angle of the solid obstacle to two different values, and the loss function equals the sum of the losses from the two states, which is defined as the difference between the actual output velocity and a prescribed desired velocity profile (Fig. 4-25 rightmost). Note that the switching angle in this design problem is also a parameter to be optimized.

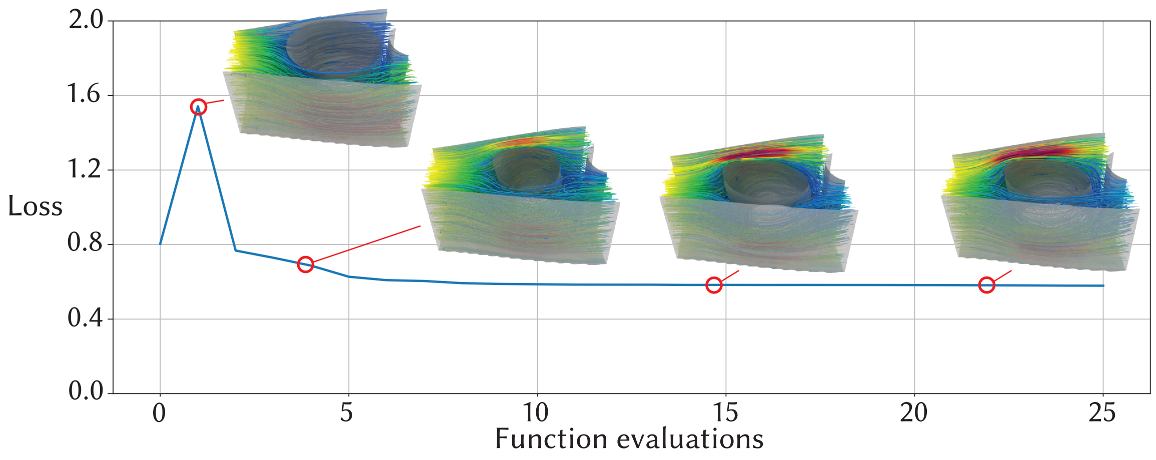


Figure 4-24: The evolution of the shape and the loss for the design of the Fluidic Switch over 25 function calls.

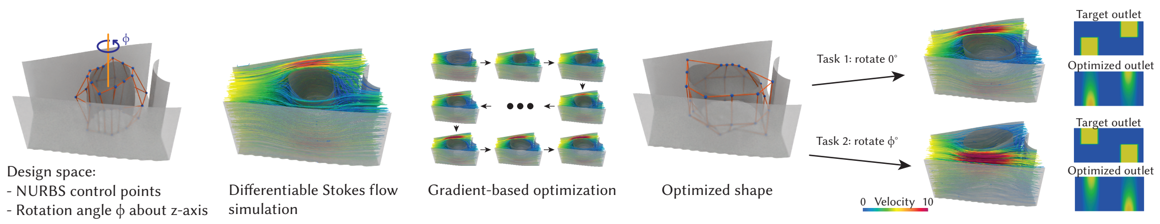


Figure 4-25: Our system automates the design of fluidic devices with differentiable stokes flow. Given a parameterized design in the form of NURBS surfaces or curves (leftmost) that separate rigid boundaries from fluid flow, we employ a Stokes flow (second from left) that evaluates the performance of this design. The flow is differentiable and gradients can be quickly evaluated, enabling gradient-based optimization (center) of the control points, and thus, the boundary. The optimized design (rightmost) can be specified to operate in one configuration or several. This example features an optimized fluidic rotational switch that shifts flow from the top outlet path to the bottom outlet path when turned.

Evaluation

Implementation details We implement our algorithm in C++ on a Linux workstation with 8 CPU cores and 32G memory. We use PARDISO [124], a parallel linear system solver for symmetric indefinite matrices, to solve the linear systems of equations in both forward simulation and gradient computation. To speed up the computation, during each function call to compute forward simulation, we prefactorize the matrix and reused the factorization in gradient computation with new right-hand sides for gradient computation. For each design problem, we start our optimization by sampling random designs and picking the best one to initialize the L-BFGS local optimization algorithm. The actual number of samples depends on the complexity of the design problem. In our experiments, we use 10 samples for problems with ≤ 10 parameters and 100 samples for larger problems. Random sampling does not create a significant time burden for our algorithm because it is easily parallelizable and requires forward simulation only. We terminate the optimizer when a maximum number of function evaluations (50 in our experiments) is reached or the solution converges into a local minimum. For all examples, we consistently observe their convergence before the maximum number of function evaluations is reached.

Performance improvement We report the statistics about the optimization process in Table 4.5 and the final designs discovered by our algorithm in Figs. 4-25, 4-20, 4-22, 4-23, and 4-24. Our algorithm improved the initial guess across the board and the final design performed significantly better than an average design (loss = 1) in all examples, with the actual improvement margin largely depending on the complexity of each problem.

It is worth mentioning that many of the novel designs revealed by our algorithm not only are physically plausible but also match the physical intuition behind the design intent. For example, in the Amplifier problem, the evolution of the boundaries narrowed the outlet so that the flow jets at a desired, faster speed (Fig. 4-20). In the Funnel example, a diagonal tunnel was formed near the end of the output in order to enforce an outlet flow whose direction is invariant to inputs (Fig. 4-22 middle). The most notable discovery of the novel design comes from the Fluidic Twister: Although the internal surface is parameterized by a NURBS surface, the final solution strongly resembles a helical surface generated by a rotated, descending ellipsoid (Fig. 4-23). The emergence of a helical surface in this design problem is no coincidence and clearly reflects the design intent of generating a swirling flow.

Linearity in the fluidic devices The KKT system in Sec. 4.2.5 connects the velocity field $\boldsymbol{v}$, the right-hand side of all boundary conditions, and the design parameters $\boldsymbol{\theta}$ in a single linear system of equations whose left-hand matrix depends on $\boldsymbol{\theta}$ only. As a result, when we fix $\boldsymbol{\theta}$, the response of the system is a linear function of the input to the system, which comes naturally from the analogy between Stokes flow and linear elasticity. Moreover, since by definition Stokes flow is steady-state, the fluidic system we investigate in this paper is therefore linear time-invariant (LTI). It is well known that the behavior of an LTI system can be fully analyzed and well understood by investigating its response to a small number of base inputs, and we made heavy use of this fact in our experiments to simplify the design problem. For example, when designing the Funnel, it is sufficient to ensure the outlet flow is diagonal under two inputs $(v_{i_1}, v_{i_2}) = (1, 0)$ and $(0, 1)$ only, and the outlet flow in response to $(0.5, 0.5)$ equals the average of the outlet flows from inputs $(1, 0)$ and $(0, 1)$ (Fig. 4-22 middle).

It is worth pointing out that while the fluidic system is LTI with a fixed $\boldsymbol{\theta}$, the full optimization problem is still highly nonlinear and non-convex. This is because $\boldsymbol{\theta}$ parameterizes the fluid-solid boundary in a nontrivial way, which is embedded in the left-hand side of the KKT system.

Boundary conditions Our convenient, explicit boundary conditions are flexible, and are a key ingredient in unlocking many of the demonstrations here. Particularly of note are the Fluidic Twister and the Fluidic Gate examples. No-separation boundaries are necessary to build up the circumferential "swirling" motion seen in the Twister example. The no-slip boundary condition, on the other hand, significantly dampens the velocities along the fluid-solid boundary, inhibiting the vortical flow. The two boundary conditions led to significantly different optimization results (Table 4.5): while a Twister optimized with a no-separation boundary achieves almost optimal performance (loss ≈ 0), optimization with a no-slip boundary hardly made any progress and performs no better than a random guess (loss ≈ 1). By contrast, the Fluidic Switch relies on no-slip boundary conditions to dampen the flow along the "off" path. If no-separation boundary conditions were to be used here, the rotational switch would need to be physically translated between configurations to completely block the "off" branches to achieve zero velocity; otherwise, some non-zero velocity will persist. We stress that our accommodation of several boundary conditions is a feature, as an engineer can achieve any of them by selecting appropriate materials along the boundary interface.

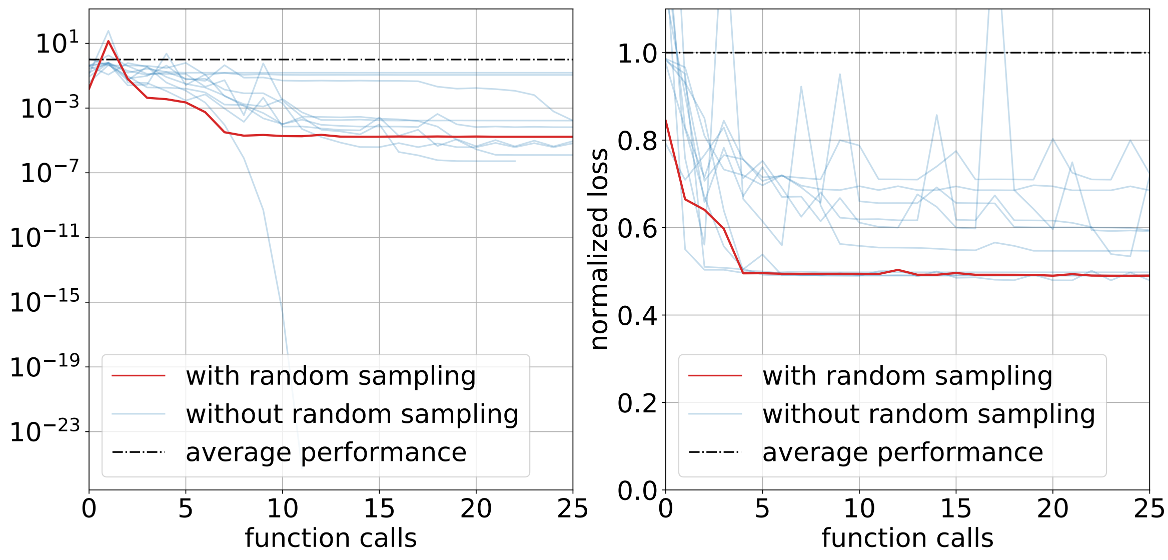


Figure 4-26: Ablation study on the necessity of global random sampling in two design problems: Amplifier (left) and Superposition Gate (right). Each blue line indicates the process of running L-BFGS optimization directly from a random design, and the red line shows our optimization progress with an initial guess from the best of 10 random designs. While a particularly good random seed can outperform our method, the flat tails from multiple random seeds reveal that local minima are common in such design problems.

Local minima Since our gradient-based optimization pipeline is inherently a local optimization method, it can suffer from getting trapped into local minima (Fig. 4-26). The distribution of local minima is problem-specific and affected heavily by the landscape of the loss function. We have partially alleviated this issue by randomly sampling multiple guesses prior to optimization and picking the best as an initial guess. While more advanced global search heuristics can be applied to our pipeline, we found this simple random sampling scheme sufficient to generate reasonably functional devices in all our design problems.

Solution convergence To prove our simulation converges under refinement and verify our governing equations approximate the truly incompressible Stokes flow, we experimented with the 2D amplifier example with an initial resolution of 32×24 cells. We then subdivided the domain by a factor of 2, 4, 8, 16, and 32, resulting in a resolution of 1024×768 eventually (Fig. 4-27 top). Additionally, we started with 32×24 cells and increased the Poisson's ratio from 0.45 until 0.499 (Fig. 4-27 bottom). We observed that in both cases, the velocity fields converged to a limit, which indicates that our quasi-incompressible Stokes flow model well approximates the truly incompressible Stokes flow.

4.2.8 Conclusion

The long-term vision of computational fluidics design is ambitious, ultimately aspiring to the automated design of complex devices such as engines, pumps, and heart valves.

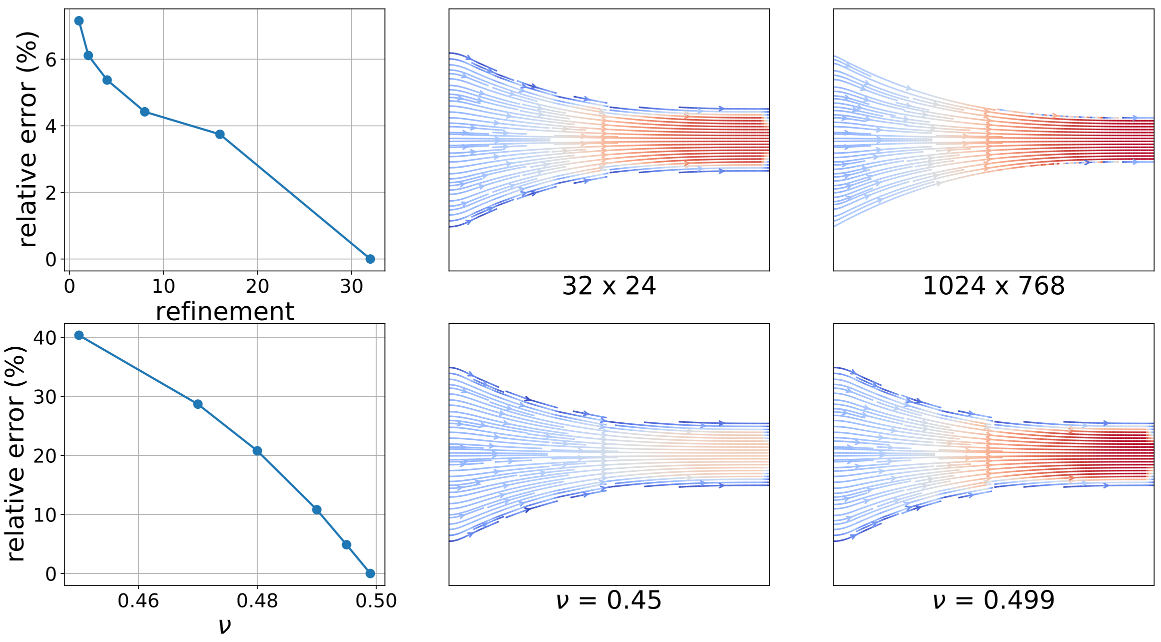


Figure 4-27: Convergence of our quasi-incompressible Stokes flow tested on the Amplifier example. Top: we solve the velocity field starting with 32×24 cells (middle) and increase the resolution by a factor of 2, 4, 8, 16, and 32 (right). The relative error (left, measured by comparing to the solution solved with $32\times$ resolution) decreases as the velocity field is solved under refinement. Bottom: we solve the velocity field with $\nu = 0.45$ (middle), 0.47, 0.48, 0.49, 0.495, and 0.499 (right). The relative error (left, measured by comparing to $\nu = 0.499$) converges to 0 as ν converges to 0.5.

Optimizing such machinery, however, is extremely challenging, requiring modeling of complex fluid dynamics while optimizing over highly complex components. We see our work as a meaningful first step toward this eventual goal. We took the Stokes flow as our fluid model, which has been used widely in engineering design and optimization problems over the past decades to model the steady-state, linearized fluidic transportation problems with a relatively low Reynolds number. Further, Stokes flow is computationally well-suited to design problems, as it is well-conditioned, linear, and provides smooth gradients, allowing for a fast inner loop of complex outer design problems. An interesting future direction to explore is to improve the expressibility of the fluid simulation method. Particularly interesting would be a steady-state Navier-Stokes fluid simulator that considers the effect of an advection term. It is also interesting to consider the effect of deformable boundaries, allowing for the design of devices with fluid-elastic coupling for applications in, e.g., soft robotics.

The second drawback of our method lies in our choice of parameterized level-sets as a design space. Such a design space was deliberately chosen as it allows for sub-grid shape design with smooth, clearly defined boundaries that separate fluid and solid regions, a common failing of topology optimization, which provides no such guarantees and only operates on the non-smooth grid cells themselves (thus making boundary conditions tricky to reason about). Still, this parameterization must be chosen by a user. A tractable method for searching both over topology while keeping boundaries smooth and regular is desired.

Third, although our Stokes solver allows for sub-grid resolution, it does not scale to arbitrarily large scenes, as it is bottlenecked by the performance of our choice of linear system solver and the optimizer. A parallel multigrid solver along with application of the adjoint method in gradient computation would allow our framework to scale to support larger problems. It would also be interesting to explore other optimizers like alternating direction method of multipliers (ADMM) [113] in our problem especially when the objective is separable on variables.

Fourth, although we purposefully kept our simulation physics-based so as to make our method amenable to real-world manufacturing, we did not fabricate and test any of our devices. Our parameterization allows engineers to specify boundaries that are physically manufacturable, without the worry for non-manufacturable parts (such as disconnected pieces in 3D). It would be interesting to physically fabricate our optimized devices and benchmark the predictive accuracy of the simulation as compared to the realized flow.

Finally, despite our initial sampling pre-processing step, whose coarse global search improves over a random starting point, there is no guarantee that our algorithm will converge to a global minimum. This is a drawback of all local continuous optimization methods like the one we employ. While smoothness of our domain helps in that we rarely find bad local minima, an algorithm for finding more globally optimal solutions is desired.

Chapter 5

Simulation-to-Reality Transfer

The last two chapters have presented a computational solution to co-designing and co-optimizing robots in simulation. While Chpt. 3 has presented a real-world application, its real-world robot design is obtained by passively executing the computational design in simulation, omitting the need to analyze and narrow the sim-to-real gap when transferring the results from simulation to reality. This chapter will fill this missing piece in the computational robot design pipeline (Fig. 5-1). Specifically, we will present a solution to the sim-to-real transfer problem using the differentiable simulator we have developed in the last chapter [44]. We demonstrate this method using an example of modeling and controlling a real-world soft underwater robot [41], which we will elaborate on below.

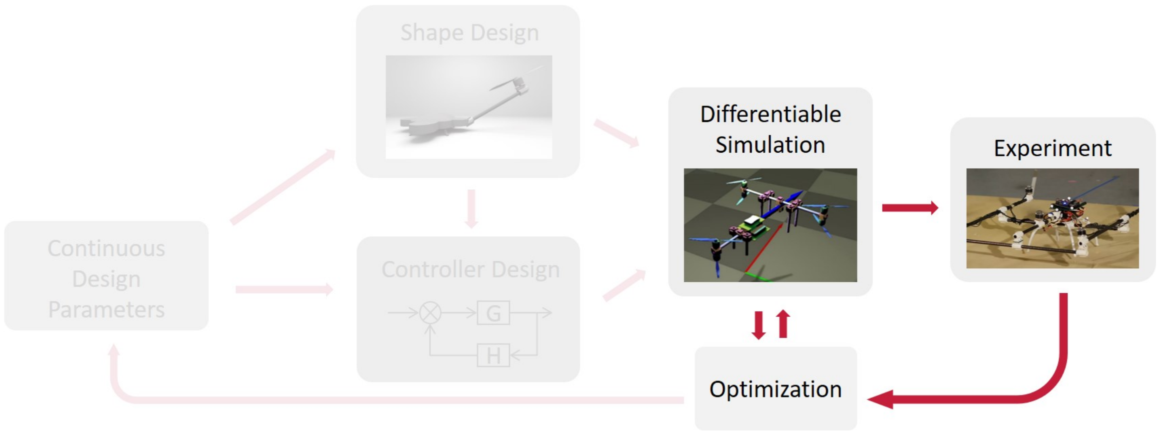


Figure 5-1: This chapter focuses on understanding and narrowing the sim-to-real gap in transferring computational design results from simulation to reality, filling the last missing piece in the computational robot design pipeline in Chpt. 1.

5.1 Motivation

Developments in marine robotics provide many advantages for tasks such as underwater exploration, sample collection, and observation of marine wildlife [19]. Aquatic animals demonstrate the advantages of having a soft-body structure for swimming and navigating aquatic environments, highlighting how compliance and flexibility is a key component for efficient underwater locomotion [114] and motivating the design of soft robotic swimmers. Although a wide variety of methods have been developed for such robots [77, 91, 129], modeling and controlling them is still an open problem due to the infinite degrees of freedom of soft systems and the problem's computational overhead.

There has been a significant body of research focusing on the modeling of soft underwater systems. This includes modeling soft-body swimmers using discrete elastic rod simulation [70] and applying the Cosserat model to Cephalopod inspired soft robots [120]. Dynamic models have also been proposed for soft fish by combining bending beam theory with hydrodynamic and damping models [46], combining beam theory with fluidic models for modeling a compliant tail [80], and modeling fish bodies as multiple compliant rigid segments with hydrodynamic forces [155]. This body of work highlights the complexity of modeling not only the deformation of the compliant robot, but also accounting for the intricate solid-fluid coupling between a robot and water. The complex dynamics of both soft robots and their interaction with their aqueous environment typically leads to a significant reality gap between simulation and real experiments. To leverage the power of simulation, such a gap must be reduced. This will allow for increasingly reliable transfer of robot controllers and designs to the real world.

In this work, we present a method for modeling and controlling underwater soft robots with a focus on narrowing the simulation-to-reality gap (Fig. 4-25). Our core idea is to embed a differentiable simulator into a pipeline that alternates between simulated and real experiments. With gradient information readily available from a differentiable simulator, previous papers have demonstrated promising results in various soft-robot applications, including system identification and controller design [44, 69, 68, 62, 16]. However, results from existing differentiable simulators are primarily focused on simulated robots, and demonstrations on real underwater soft robots have yet to be seen. Our work attempts to fill this gap by coupling differentiable simulation with a differentiable, analytical hydrodynamic model, to enable improved modeling and optimization of water-based soft systems.

Our pipeline starts by collecting motion data from a real underwater soft robot with synthetic control signals (Fig. 4-25, left). A dynamic model of the soft robot, including its actuators and its hydrodynamic forces, is initialized in simulation. The initial values of the model parameters are obtained from measurements on the soft robot and estimation from previous papers. Next, our pipeline compares the collected data and the motion predicted by the dynamic model in simulation, and gradients of the model parameters with respect to the error between the model and real world are automatically computed by the simulator to reduce the modeling error (Fig. 4-25, lower right). With an improved dynamic model in simulation, the pipeline then runs trajectory optimization to propose a new open-loop controller (Fig. 4-25, upper right), which is then executed on the hardware platform to collect more data for the next iteration (Fig. 4-25, left). The output of our pipeline is a calibrated dynamic model and an optimized open-loop controller that can be directly used on a real robot.

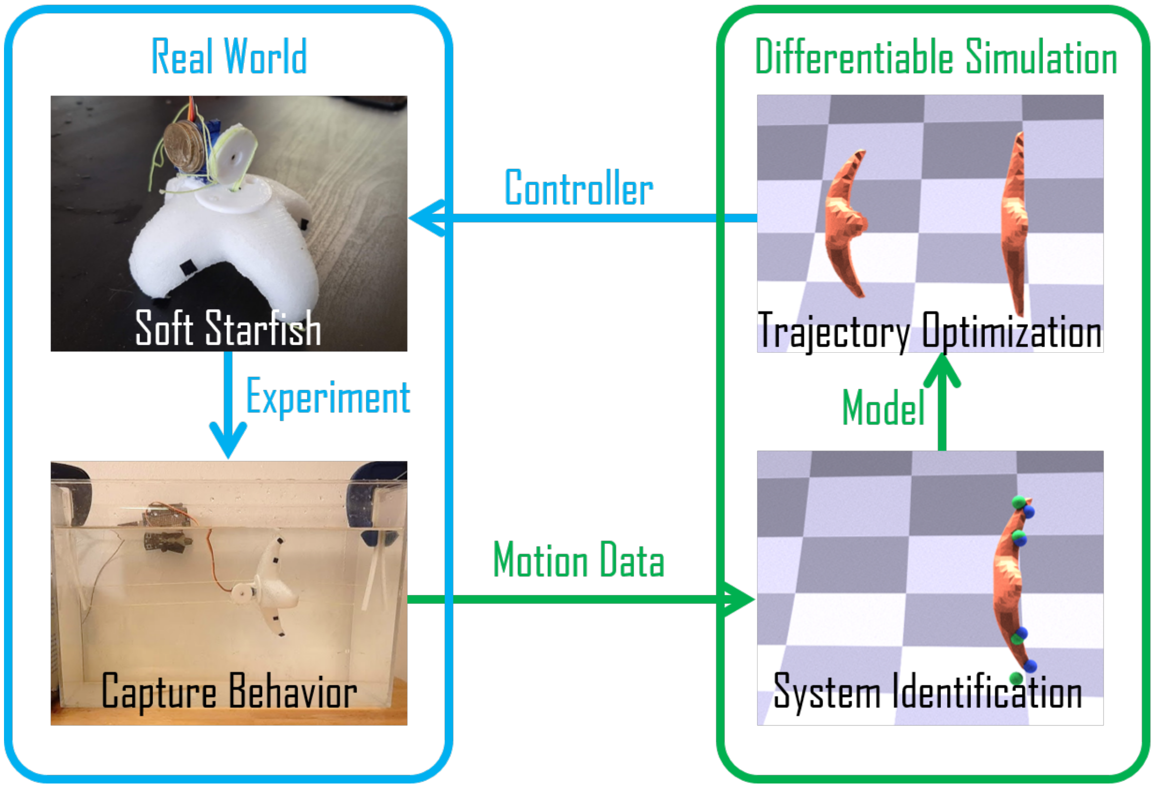


Figure 5-2: An overview of the iterative, real-to-sim pipeline. Real-world motion data is captured from the robot for a given control input (left). This data is transferred to the differentiable simulator where system identification narrows the gap between reality (four green spheres bottom right) and simulation (four blue spheres bottom right) after which trajectory optimization is used to generate a new control sequence. Through iterative transfer we show trajectory optimization and reduction of the reality gap.

We demonstrate the efficacy of our pipeline on Starfish, a customized underwater soft robot design made of silicone foam and actuated with four tendons. Like many other existing soft robot designs, Starfish leverages non-symmetric placement of tendons or wires to achieve bending. Similar approaches have been shown to be effective in soft jellyfish robots [149, 3], and for undulating fish swimmers [164] including micro robot swimming fish [157]. Starfish is connected to a rail in a water tank to limit its motion to horizontal motion only. We find that our pipeline manages to not only narrow down the simulation to reality gap but also produce an effective open-loop controller after a few iterations, whose performance is increased significantly when compared to a handcrafted baseline controller.

5.2 Simulation

We now describe our simulation model for Starfish as well as its implementation in a differentiable simulator. We choose to base our simulator implementation on

DiffPD [44] and augment DiffPD by implementing a differentiable actuator and hydrodynamic model with trainable parameters in optimization. It is worth mentioning that our pipeline is agnostic to the choice of differentiable simulators.

5.2.1 Governing Equations

We model the body of Starfish using the finite element method (FEM) with a tetrahedral discretization. An implicit Euler time-stepping scheme is used because of its numerical robustness and large time steps. Let n be the number of nodes after discretization and let $\mathbf{x}_i \in \mathbb{R}^{3n}$ and $\mathbf{v}_i \in \mathbb{R}^{3n}$ be the nodal positions and velocities at the i-th time step. The governing equations can be written as follows:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_{i+1}, \tag{5.1}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \frac{h}{m}[\mathbf{f}_e(\mathbf{x}_{i+1}) + \mathbf{f}_h(\mathbf{x}_i, \mathbf{v}_i) + \mathbf{f}_a(\mathbf{x}_{i+1}, \mathbf{a}_i)]. \tag{5.2}$$

Here, h is the time step (1/60 second in our simulation), m the mass of a node, $\mathbf{f}_e$ the elastic force computed from the material model, $\mathbf{f}_h$ the hydrodynamic force, and $\mathbf{f}_a$ the actuation force dependent on the action $\mathbf{a}_i$ at this time step. Eqns. (5.1) and (5.2) contain parameters whose values need to be determined from the real system to build an accurate dynamic model. These include the material parameters in $\mathbf{f}_e$, the hydrodynamic parameters in $\mathbf{f}_h$, and the actuator parameters in $\mathbf{f}_a$, which are described in detail in the following three subsections.

5.2.2 Material Model

We use the same material model as described in DiffPD [44] and Min et al. [107], which is based on the corotated linear material model [130]. The material model has three parameters that require either a direct measurement or a reliable estimation: its density, its Young's modulus, and its Poisson's ratio. We compare these parameters to the product specification [134] of the silicon foam used (SomaFoam 25) to identify its density (400kg/m^3) and Poisson's ratio (0.48) reliably. The Young's modulus E is not included in the product specification, so we estimate its value based on the reported shore hardness and the Gent's relation [57]. Since this is an empirical estimation with a possibly large uncertainty, we make E a trainable parameter in our optimization.

5.2.3 Hydrodynamic Forces

We model hydrodynamic effects as external and explicit forces added to the system. For the sake of speed and simplicity in gradient computation, we choose to compute thrust and drag from water with a widely used approximation in aerodynamics [162, 44, 107]:

$$\mathbf{f}_d = \frac{1}{2}\rho A C_d(\alpha)\|\mathbf{v}_{rel}\|_2 \mathbf{v}_{rel}, \tag{5.3}$$

$$\mathbf{f}_t = -\frac{1}{2}\rho A C_t(\alpha)\|\mathbf{v}_{rel}\|_2^2 \mathbf{n}. \tag{5.4}$$

Here, $\mathbf{f}_d$ and $\mathbf{f}_t$ represent the drag and thrust forces evaluated on every triangle on the surface of Starfish after discretization. We use ρ to represent the density of water and A the area of the triangle. $\mathbf{v}_{rel} \in \mathbb{R}^3$ is the relative velocity between Starfish and the flow of water, and α is the angle of attack. C_d and C_t are scalar functions computing the drag and thrust coefficients. For aerodynamic applications, C_d and C_t are typically measured by conducting wind tunnel experiments. For our underwater experiments, however, a direct measurement of C_d and C_t is difficult. Therefore, we represent C_d and C_t as B-splines and make their control points trainable in our optimization. We initialize the B-splines with the curves suggested in Min et al. [107] and optimize their shapes with the gradients calculated in the differentiable simulator.

5.2.4 Actuators

As we use tendons inside foam to actuate Starfish, we model the actuation with an anisotropic elastic energy which exerts large forces along the tendon direction [44, 107]. For a tetrahedron through which the tendon passes, the associated anisotropic elastic energy is defined as follows:

$$E_a = \frac{w}{2}\|(1 - a_i)\mathbf{F}\mathbf{m}\|_2^2, \tag{5.5}$$

where w is a prespecified stiffness, $\mathbf{F}$ is the deformation gradient, $\mathbf{m}$ is the direction of the tendon, and $a_i \in [0, 1]$ is the control signal. Smaller a_i indicates greater contraction along the tendon direction. The actuation force $\mathbf{f}_a$ for each node is then computed by aggregating E_a from its adjacent tetrahedrons and calculating its spatial gradients. Our actuator model has one trainable parameter w in our optimization.

5.3 Optimization

Our ultimate goal is to find an open-loop controller for Starfish that maximizes its forward velocity. In this section, we describe how we combine the simulation model and real motion data to achieve this goal. Our core idea is to leverage the gradients from the differentiable simulator to improve both the dynamic model (i.e., system identification) and the open-loop control signals via trajectory optimization.

5.3.1 Problem Definition

We abstract the simulation model in Sec. 5.2 as follows:

$$\mathbf{s}_{i+1} = \mathbf{DiffSim}(\mathbf{s}_i, \mathbf{a}_i; \theta) \tag{5.6}$$

where $\mathbf{s}_i = (\mathbf{x}_i, \mathbf{v}_i)$ represents the state of the robot at the i-th time step, $\mathbf{a}_i$ is the actuation signal as described before, θ represents model parameters, and **DiffSim** can be any black-box differentiable simulator that computes $\mathbf{s}_{i+1}$, the new state of the system after one time step. The model parameters θ consists of all trainable parameters in Sec. 5.2, which we summarize in Table 5.1.

The decision variables to be optimized in our problem are the model parameters θ and the sequence of actions $\mathbf{a}_i$. For optimizing θ, we consider minimizing the following objective L_θ:

$$\min_{\theta} \quad L_\theta. \tag{5.7}$$

$$s.t. \quad L_\theta = \sum_i \|\mathbf{s}_i - \mathbf{s}_i^*\|^2, \tag{5.8}$$

$$\mathbf{s}_{i+1} = \mathbf{DiffSim}(\mathbf{s}_i, \mathbf{a}_i^*; \theta), \quad \mathbf{s}_0 = \mathbf{s}_0^*, \tag{5.9}$$

where $\mathbf{s}_i^*$ and $\mathbf{a}_i^*$ refer to the state and the action signal from the measurement at the i-th time step. In short, we adjust θ to match the motion of Starfish in reality to its counterpart in simulation.

The objective for optimizing $\mathbf{a}_i$ is defined as $L_\mathbf{a}$ below:

$$\min_{\mathbf{a}_i} \quad L_\mathbf{a}. \tag{5.10}$$

$$s.t. \quad L_\mathbf{a} = COM(\mathbf{s}_N) - COM(\mathbf{s}_0), \tag{5.11}$$

$$\mathbf{s}_{i+1} = \mathbf{DiffSim}(\mathbf{s}_i, \mathbf{a}_i; \theta), \quad \mathbf{s}_0 = \mathbf{s}_0^*. \tag{5.12}$$

where N is the index of the last time step considered in this trajectory optimization problem and $COM(\mathbf{s})$ computes the x coordinate of the center of mass from state $\mathbf{s}$. We define the center of mass as the average of all vertices from $\mathbf{s}$. Since the forward direction of our Starfish is along the negative x axis, minimizing $L_\mathbf{a}$ will maximize the traveling distance as desired. Note that θ is fixed in this optimization problem.

For a standard differentiable soft-body simulator, the procedure of computing the gradients with respect to θ and $\mathbf{a}_i$ is well documented in previous work [69], which interested readers can refer to for more details. We use L-BFGS, a gradient-based quasi-Newton method to solve the two optimization problems above.

5.3.2 Optimization Algorithm

We now present an alternating scheme to improve both the model parameter θ and the sequence of actions $\mathbf{a}_i$. Our optimization process starts with an initial guess of θ (Table 5.1) and $\mathbf{a}_i$ (a synthetic control signal). We execute $\mathbf{a}_i$ on the hardware and collect the measurement $(\mathbf{s}_i^*, \mathbf{a}_i^*)$. Note that $\mathbf{a}_i^*$ and $\mathbf{a}_i$ differ slightly because the motor quantizes the real number $\mathbf{a}_i$ into integers. We then use $\mathbf{a}_i^*$ and $\mathbf{s}_i^*$ to minimize Eqn. (5.7) and obtain an improved model parameter θ. Next, with the optimized

Table 5.1: A summary of all trainable model parameters.

Name	Definition	Initial guess
E	Young's modulus	0.9MPa, estimated from [134].
w	The actuator's stiffness	2MPa, from [44].
P_{C_d}	Four control points of C_d	From [107], Fig. 2.
P_{C_t}	Four control points of C_t	From [107], Fig. 2.

dynamic model, we run trajectory optimization to update $\mathbf{a}_i$. Finally, we test $\mathbf{a}_i$ on the hardware, initiating the next round of experiment and closing the optimization loop in our pipeline. Alg. 7 summarize the whole optimization algorithm in pseudo-code.

Algorithm 7: Co-optimize model (θ) and actions ($\mathbf{a}_i$)

```
Input: Initial $\theta$ and $\mathbf{a}_i$;
Output: Optimized $\theta$ and $\mathbf{a}_i$;
while experiments do not converge do
    // Hardware experiment;
    Execute $\mathbf{a}_i$ on the hardware to collect $\mathbf{s}_i^*$ and $\mathbf{a}_i^*$;
    // Check convergence;
    Use $\mathbf{s}_N^*$ and $\mathbf{s}_0^*$ to compute $L_{\mathbf{a}_i}$;
    if $L_{\mathbf{a}_i}$ is similar to the last iteration then
        // Convergence;
        break;
    // System identification;
    Minimize $L_\theta$ (Eqn. (5.7)) to update $\theta$;
    // Trajectory optimization;
    Minimize $L_\mathbf{a}$ (Eqn. (5.10)) to update $\mathbf{a}_i$;
```

5.4 Results and Discussions

5.4.1 Hardware Setup

Fabrication The fabrication method has been chosen to allow for transfer from a 3D CAD model to a real-world robot while minimizing the "fabrication gap" between the real and simulated system. Starfish is fabricated by creating an inverse mould into which silicone foam (SomaFoam 25, SmoothOn) is cast. Silicone foam, a material widely used for soft robotic fabrication [135], has been chosen as it allows for rapid fabrication, shows elastic properties, and has natural buoyancy. The "muscle fibers" or tendons can then be routed into the soft structure along the bottom of each of the legs of Starfish. The tendons are inserted using a thin metal tube through which the tendon fibers (non-extensible fishing line) can be inserted, and the tube removed. Each tendon is fixed at the end of each leg using adhesive and connected to a servo motor via an incompressible tube which runs through the center of the body of Starfish where they are connected to the servo by a pulley. The servo can contract the tendons, flexing the legs inwards, and then extend the tendons, to flex the legs to their initial position. The motion is highlighted in Fig. 5-3. It is important that the tendon length can be set accurately for effective sim-to-real transfer, as such the servo was chosen to have a torque which is higher than the load. This was validated by performing no-load and load tests and observing that the servo position is not significantly affected by the load. The servo position is controlled by a microcontroller which sets the position via a PWM signal.

Experimental setup A tank-based experimental setup has been created for testing the robot. To constrain the robot in an orientation that allows for motion capture, the tank system has been fitted with horizontal rails constructed from fishing twine. Starfish has low friction PTFE tubing through the body through which the guide rails run. The use of low friction materials and the presence of water results in the rails exhibiting low friction and enabling the robot to move forwards while the orientation is fixed. The rails do provide some negating frictional force potentially reducing the

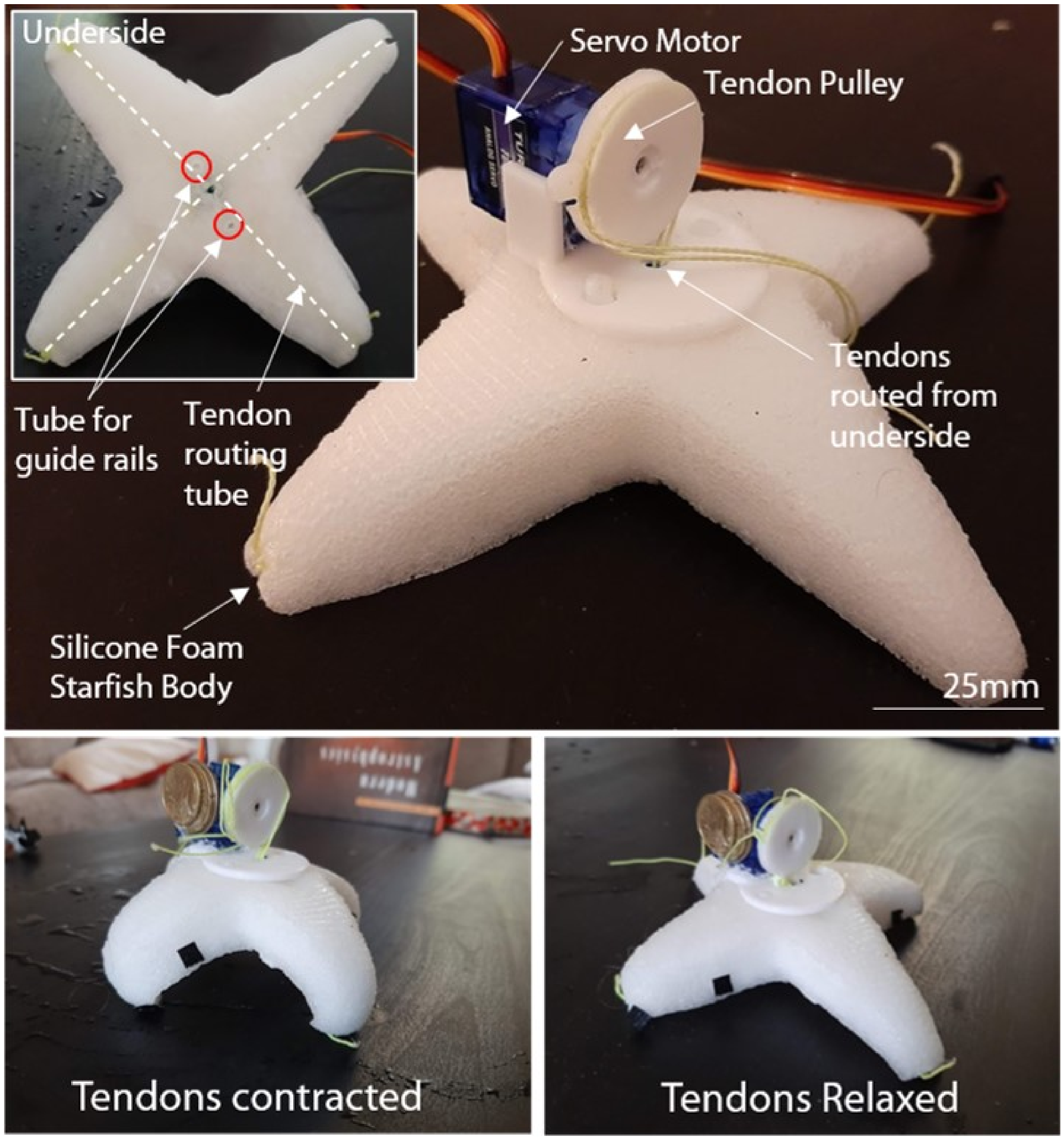


Figure 5-3: The fabricated soft robot (top) showing the servo-based mechanism and the inset figure showing the underside and the tendon routing. The contracted and relaxed pose of the robot are shown in the bottom pictures.

forward velocity. However, we expect this to be minimal. The weight of the robot has been adjusted such that it has approximately neutral buoyancy at the depth the rails are within the tank. Fig. 5-4 shows the experimental setup.

To capture motion data from the soft robot, a high-speed camera (Logitech BRIO) has been fixed outside the tank. Four black markers were attached to one side of Starfish at locations which capture the most dynamic information about the robot. To capture the 2D motion data from these markers, the video feed was calibrated using a standard checkerboard and the marker locations extracted by tracking features corresponding to the makers throughout the video.

In each experiment, the motion of the robot and the control sequence (i.e., the length of contraction of the tendon) is recorded at 60Hz.

5.4.2 Experimental Verification

To verify the experimental setup and ensure that the interaction between the fluid and Starfish is the direct cause of any resultant forward motion, we show in Fig. 5-5 the motion of Starfish when the tank is both full of water and empty with a given cyclic sequence of actions. The fact that the robot barely moved without water shows the influence of solid-fluid interaction and the necessity of calibrating the hydrodynamic model.

In addition to verifying the experimental setup, the repeatability and reliability of

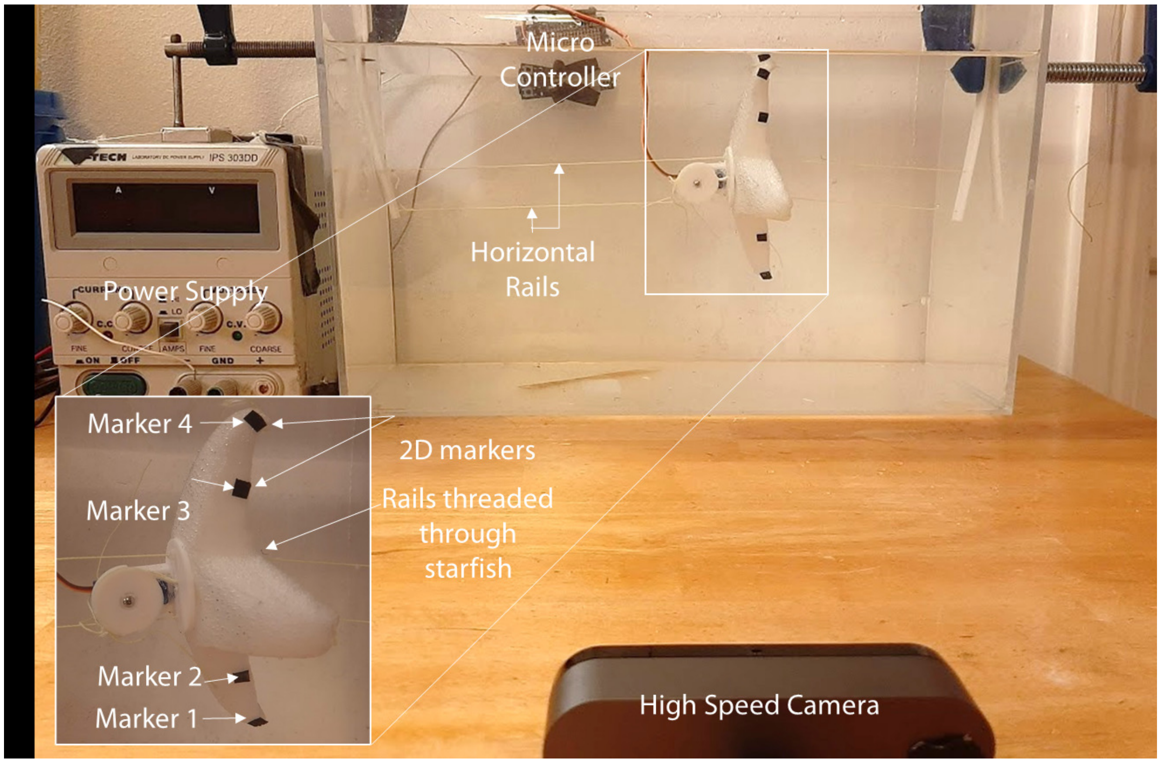


Figure 5-4: Experimental setup showing the tank with the horizontal rails and the robot. The high-speed camera and markers allow the motion of the robot to be captured.

the experiments must be demonstrated to show that experiments are representative. To show this, we ran the robot with a cyclic sequence of actions and observed cyclic motions were established after the initial transient state of water (Fig. 5-5), indicating that repeated control signals lead to reproducible motions in our experiments.

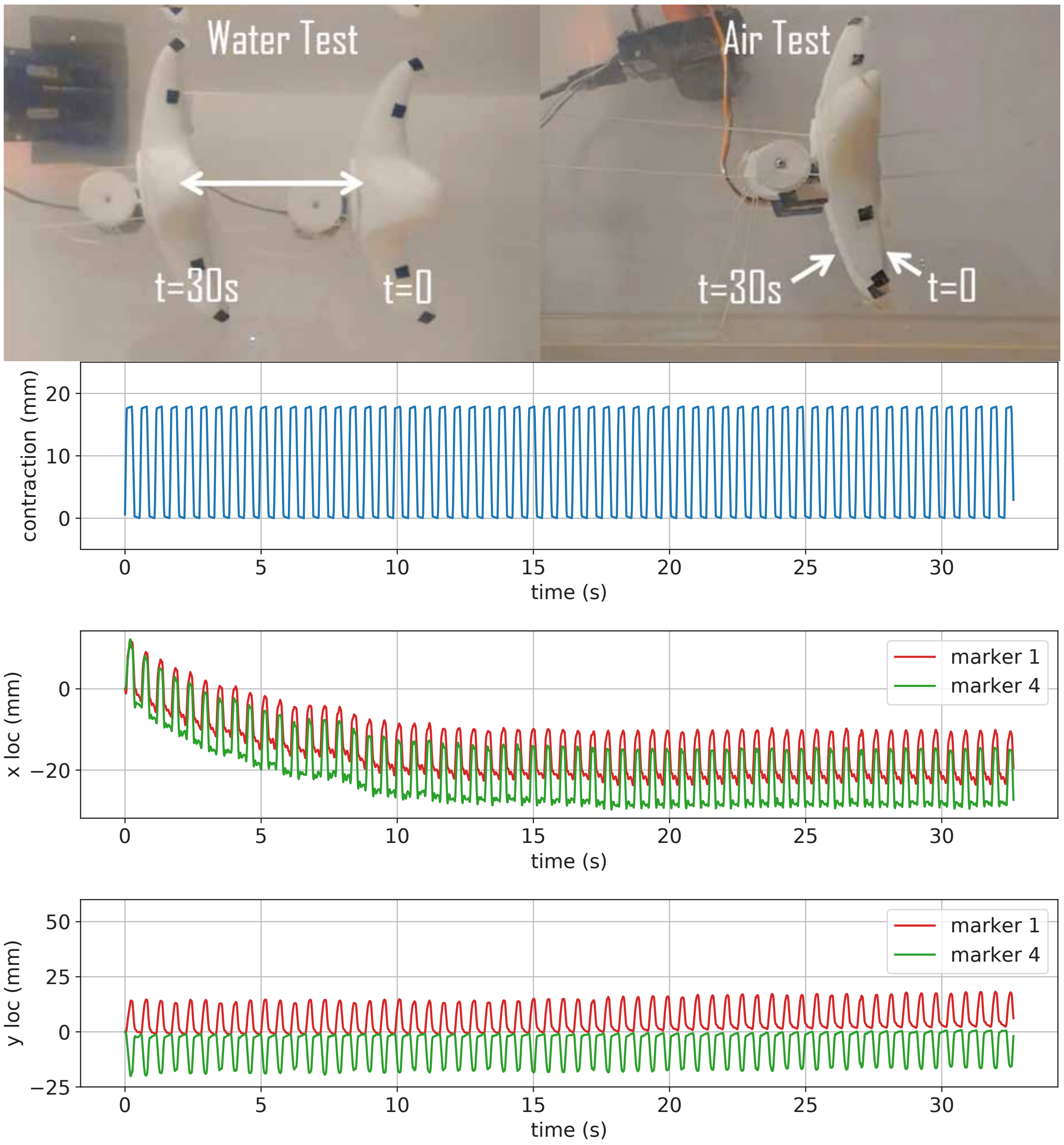


Figure 5-5: Top row: The trajectories of Starfish running the same cyclic control sequence with water (left) and without water (right). The robot made little progress when water was not present. Bottom three: A cyclical control input (upper middle) is applied to the robot, with the outer most markers (marker 1 and marker 4) positions recorded. The tracked horizontal marker position (lower middle) and vertical marker position (bottom) show that after some initial transients, repeatable and cyclical movement is achieved.

5.4.3 Baseline Algorithms

To better evaluate the performance of our algorithm, we propose two baselines for comparison: **bl-ctrl** and **bl-one-iter**. Both baselines provide an open-loop controller that attempts to maximize the traveling distance of Starfish, which is our ultimate goal in physical experiments. The **bl-ctrl** baseline proposes to use a sinusoidal sequence of actions with an educated guess on its frequency and amplitude without further optimization. The role of this baseline is to understand if the problem can be solved trivially by a carefully chosen handcrafted solution. The **bl-one-iter** baseline simply runs our pipeline for 1 iteration and terminates, i.e., it conducts system identification and optimizes $\mathbf{a}_i$ exactly once. Comparing our pipeline to **bl-one-iter** will evaluate the necessity of alternating between system identification and trajectory optimization for multiple iterations. To ensure a fair comparison, we use **bl-ctrl** as the initial guess of $\mathbf{a}_i$ in both **bl-one-iter** and our pipeline (Alg. 7).

5.4.4 Optimization Results

We now report the progress of our optimization pipeline and the performance of the optimized controller both in simulation and in reality. Note that the progress of our pipeline also covers the performance of the two baselines above. This is because **bl-ctrl** and **bl-one-iter** can be interpreted as terminating our pipeline at the beginning and after 1 iteration, respectively. We optimize a 3-second-long sequence of action in simulation and test it on Starfish for 30 seconds by repeating the sequence 10 times. Table 5.2 summarizes the system identification loss L_θ, the trajectory optimization loss $L_\mathbf{a}$, and the average velocity of the robot in the simulation environments (v_s) and the physical experiments (v_r). At each iteration, lower L_θ and $L_\mathbf{a}$ and higher v_r are better. We have also reported the optimized Young's modulus E, the actuator stiffness w, and the control points of C_d and C_t at each iteration in Table 5.2. To visualize the optimized results, we render the motion of the simulated robot before and after each iteration's optimization in Fig. 5-6 and plot the optimized control signals in Fig. 5-8. Finally, we show the performance of our optimized open-loop controller in real-world experiments in Fig. 5-9.

5.4.5 Discussion

Comparisons with baselines By comparing the quantitative results between different methods in Table 5.2, we reach the following conclusions: First, the control signal proposed by **bl-ctrl** performs poorly without further system identification or trajectory optimization. The real robot travelled at $0.21 cm\, s^{-1}$, corresponding to only 6 centimeters during the 30-second-long test time. This shows that finding an open-loop controller for an underwater soft robot is not a trivial task. Second, and more importantly, we noticed that the traveling velocity measured in real experiments increases monotonically with each iteration until the optimization process converges. After 1 iteration, the traveling velocity from **bl-one-iter** ($0.48 cm\, s^{-1}$) is more than twice that of **bl-ctrl** ($0.21 cm\, s^{-1}$). This trend continues until the algorithm convergences after three more iterations with a velocity of $0.75 cm\, s^{-1}$. Such an improvement after each iteration highlights the value of running Alg. 7 for multiple iterations.

Sim-to-real gap Another metric of success in our experiments is whether the sim-to-real gap has been narrowed after optimization. The sim-to-real gap measures the discrepancy between the dynamic model in simulation and the robot in real experiments, which can be understood by answering two questions: First, does the dynamic model fit the given measurement data well? Second, can the model predict new behaviors accurately? Such questions can also be motivated from the classic bias-variance tradeoff in machine learning, which aims to explain the expressiveness and generalizability of a model.

To answer the first question, we refer readers to the second column of Fig. 5-6. By definition, the distance between the measured and simulated marker positions (green and blue spheres respectively) is a direct, quantitative metric of the fitting error of our model (see also the L_θ column in Table 5.2). By comparing column 1 and 2, we can see that our system identification step manages to explain the measurement data well, as indicated by the closer distance between blue and green spheres after optimization.

To answer the second question, i.e., the generalizability of our dynamic model after system identification, we compare the simulated and actual motions of the robot

Table 5.2: The optimization progress of Alg. 7

Iter.	L_θ	$L_\mathbf{a}$	v_s $(cm\, s^{-1})$	v_r $(cm\, s^{-1})$	E	w
0 (**bl-ctrl**)	9.2e-2	-1.2e-2	0.01	0.21	9.0e5	2.0e6
1 (**bl-one-iter**)	2.9e-2	-3.8e-2	0.83	0.48	5.0e5	4.1e6
2	7.7e-2	-3.4e-2	0.68	0.56	1.0e6	1.4e6
3	5.1e-2	-3.5e-2	0.66	0.67	4.3e5	4.8e6
4	5.2e-2	-3.9e-2	0.77	0.75	4.0e5	5.7e6
5	7.5e-2	-3.9e-2	0.75	0.75	3.8e5	5.8e6
Iter.	$P^1_{C_d}$	$P^2_{C_d}$	$P^3_{C_d}$	$P^4_{C_d}$		
0	0.1	0.1	1.9	2.1		
1	0.0	0.0	0.5	2.0		
2	0.2	0.2	0.7	2.4		
3	0.0	0.0	0.9	2.5		
4	0.0	0.0	0.8	2.2		
5	0.0	0.0	0.8	2.2		
Iter.	$P^1_{C_t}$	$P^2_{C_t}$	$P^3_{C_t}$	$P^4_{C_t}$		
0	-0.8	-0.5	0.1	2.5		
1	-0.5	0.0	1.0	3.0		
2	-0.5	-0.2	0.7	3.0		
3	-0.6	-0.5	0.0	3.0		
4	-0.6	-0.2	0.5	3.0		
5	-0.6	-0.2	0.5	3.0		

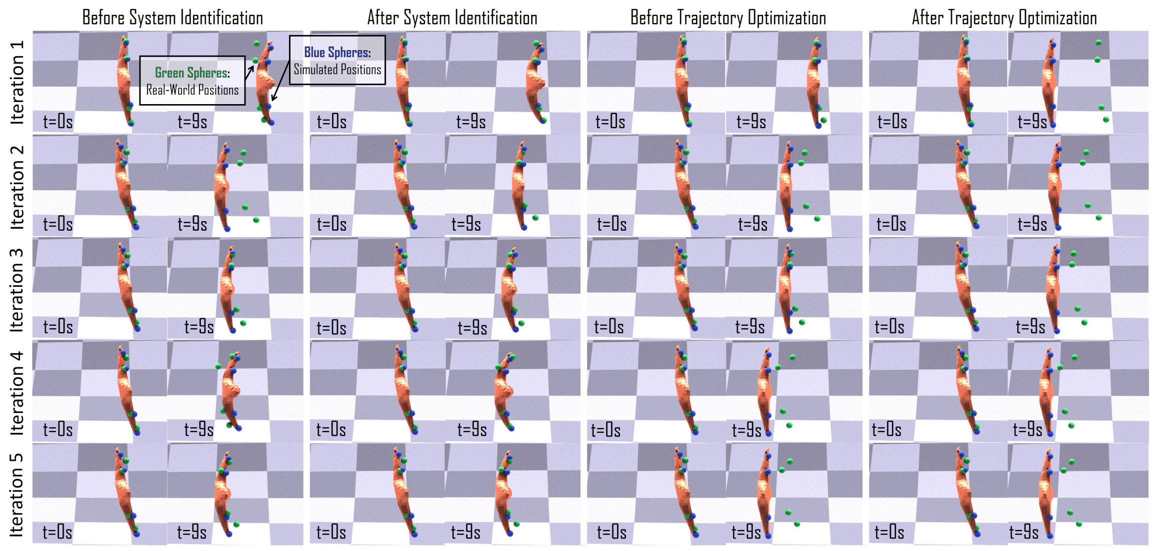

Before System Identification
After System Identification
Before Trajectory Optimization
After Trajectory Optimization
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Green Spheres: Real-World Positions
Blue Spheres: Simulated Positions
t=0s
t=9s

Figure 5-6: Left two columns: the motion of our simulated Starfish before (column 1) and after system identification (column 2) for the 1st to 5th iterations. The blue and green spheres indicate the 4 marker's locations in simulation and from real experiments, respectively. We visualize the motion of the robot at $t = 0$ and $t = 9$ seconds. The goal of this optimization is to narrow the distance between each pair of blue and green spheres. Right two columns: the motion before (column 3) and after trajectory optimization (column 4). The goal of this optimization is to push the robot towards its left as much as possible. Note that we assume the motion to be solved in trajectory optimization is cyclic with a period of 3 seconds.

with a sequence of action *not seen* in the training process of the dynamic model. Such a comparison is reflected in the first column of Fig. 5-6, i.e., the simulated and actual motions *before* system identification at each iteration. Note that the simulated and actual motions in this column execute the same sequence of action, but the corresponding motion capture data have not been used to train the dynamic model yet (which is what the algorithm is about to do afterwards). This also serves as a direct measurement of the reality gap, i.e., if we run the simulated and actual robot with exactly the same control signal, how different could the motions be? To quantify the motion difference, we report in Table 5.2 the average velocity from this motion in simulation (the v_s column) and the real experiments (the v_r column). It can be seen that the difference between v_s and v_r becomes significantly smaller as the algorithm proceeds with more iterations. Specifically, the two velocities become almost identical after only 3 iterations, indicating our algorithm's good generalizability as well as its effort into narrowing the sim-to-real gap. To visualize the motions more thoroughly, we plot the location of the robot's center of mass in these two motions obtained at the last iteration of our algorithm in Fig. 5-7. It can be seen that although the absolute location of the center of mass still differs from time to time between simulation and reality, the simulated motion exhibits local, oscillating patterns that are very similar to its real-world counterpart. The full motion sequences can be found in our video and in Fig. 5-9.

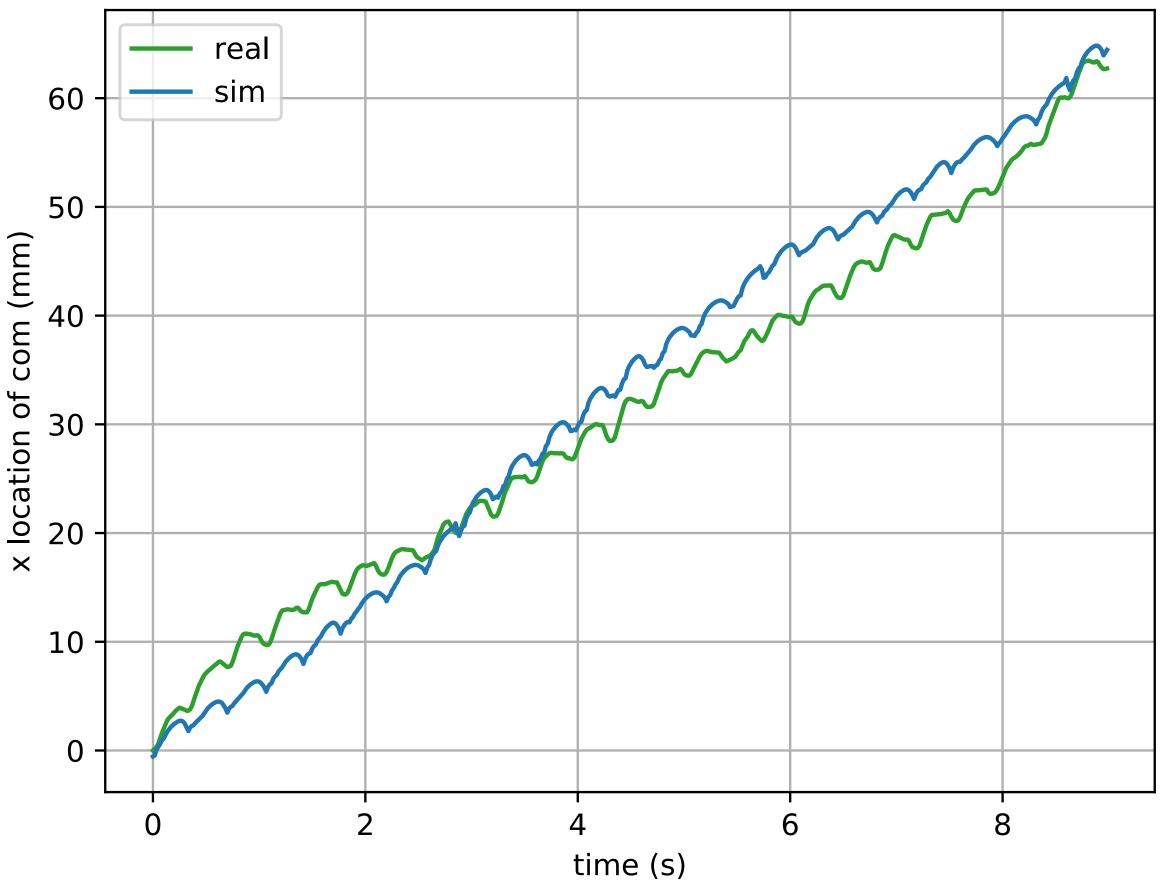


Figure 5-7: The motion of the robot's center of mass in simulation and in the real experiments with the identical control signal. The simulation data is computed with the dynamic model after our algorithm converges.

Optimized controller To better understand the optimized control sequence, we plot the intermediate controllers after each iteration in Fig. 5-8. Comparing to the baseline controller proposed in **bl-ctrl**, we notice the optimizer made two significant changes to the control sequence: First, it increased its amplitude by about 16% (from 12mm to 14mm). Second, it injects very high-frequency signals from time to time. We believe that both changes allow Starfish to leverage hydrodynamic forces more effectively and lead to the longer traveling distance.

5.5 Conclusion

Computational tools for dynamic modeling and controller development of soft robotics have the potential to change how we design and control soft robots. However, the modeling of soft structures and environmental interactions make this challenging. Differentiable simulators offer potential advantages as they expose gradient information and also show computation efficiency. We proposed a pipeline for the development of an open-loop controller for a swimming soft robot using a differentiable simulator

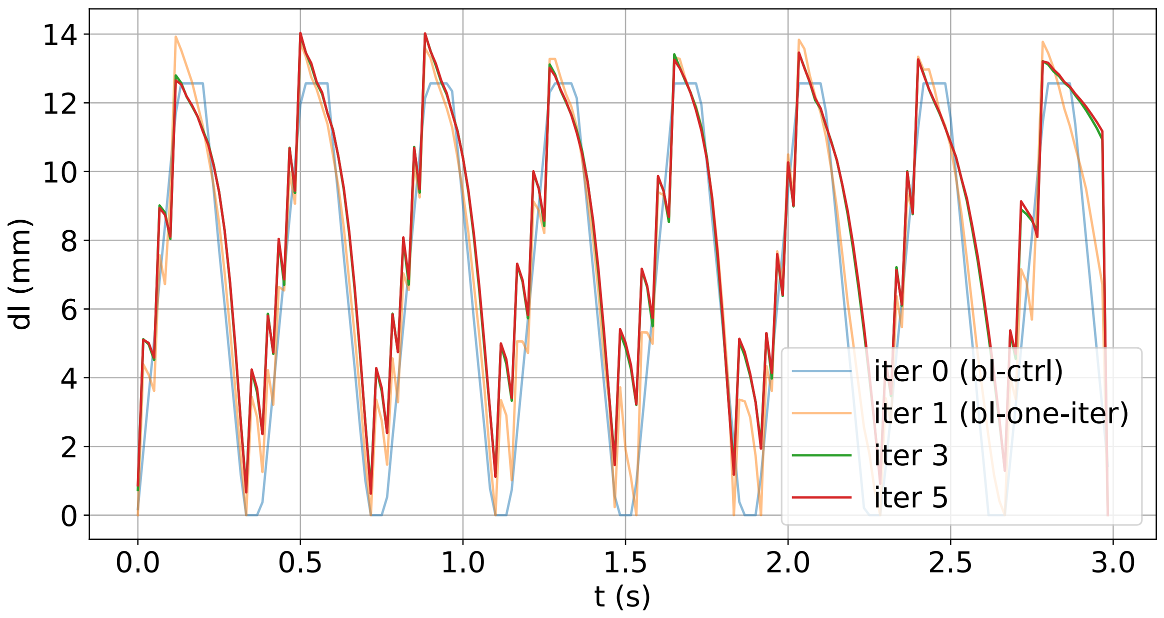


Figure 5-8: The optimized control sequences $\mathbf{a}_i$, reported as the tendon contraction (dl in millimeters) from our method and two baselines.

in which we iteratively loop between the real and simulated worlds. We demonstrate this approach on a simple four-legged starfish-shaped soft swimming robot. Within four iterations, we show the forward swimming velocity can be increased by a factor of 3.6 in comparison with a handcrafted baseline. In addition, we show that the sim-to-real gap, with the simulation showing realistic dynamic behaviors.

While this approach demonstrates how the simulation-to-reality gap can be *qualitatively* reduced, there still remains a *quantitative* gap, as can be seen from the discrepancy between the simulated and actual center-of-mass motions in Fig. 5-7. We believe this is due to inaccuracies in hydrodynamic force and actuator modeling. The pipeline concept we propose can be used interchangeably with alternative simulators and other robots. This could allow for simulators with alternative hydrodynamic models to be explored as a means as further reducing the sim-to-real gap. However, despite this, we still demonstrate how the performance of the robot can be improved, showing that despite the existence of a “quantitative reality gap”, by reducing the “qualitative reality” gap, optimization is still successful and contributes to the performance improvement observed in real experiments.

In this first demonstration of the iterative use of a differentiable simulation for system identification and trajectory optimization, we have considered a relatively simple robot. Our Starfish has only a single control signal for all four limbs. For robots with an increased number of actuators, the system identification and control problem becomes more challenging. The iterative approach we propose may become increasingly beneficial as it allows for a wider range of conditions to be experienced which otherwise may not be observed. This is a trade-off with the additional complexity. Future work should investigate how the pipeline can be optimized by considering control sequences not only optimized for forwards locomotion but also system identification.

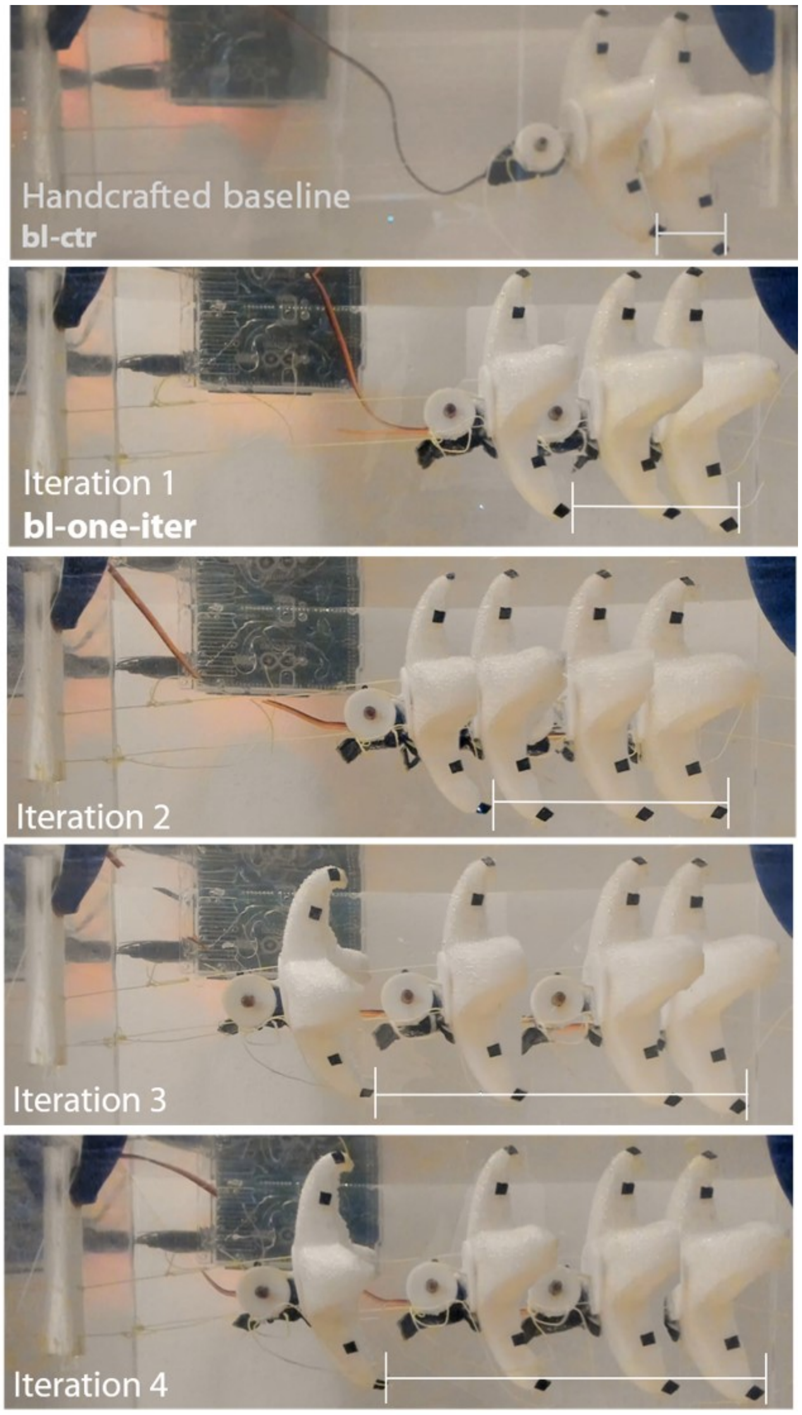


Figure 5-9: Overlaid motion sequence showing the progress made by the robot in the fixed time period (30 seconds) when using the baseline handcrafted control sequence (top), the **bl-one-iter** baseline control sequence (upper middle), and control sequences for the second to fourth iteration.

Chapter 6

Conclusions, Limitations, and Future Work

As of today, co-designing the body and brain of a robot in the real world is still a challenging problem due to the intricate coupling between geometry, control, simulation, and fabrication. The process of discovering the best robot for a given task is still largely driven by human experts, making this process labor-intensive and error-prone. While engineers do exploit computational tools, e.g., CAD software and physics simulators, to assist themselves, we argue in this thesis that these computational tools have much more potential and deserve more attention today than before. With carefully designed computational methods in geometrical design, control, simulation, and optimization discussed in this thesis, we envision a holistic, fully automatic computational design process of robots that can reveal novel and extreme designs beyond the knowledge boundary of human experts.

There are still quite a few limitations in the computational design pipeline and differentiable simulation methods discussed in this thesis. These limitations lead to many exciting research problems in designing, controlling, simulating, optimizing, and fabricating robots. Below, we suggest a few future research directions that we consider promising based on our preliminary exploration. We believe breakthroughs along these directions will significantly push the frontier of computational robot design.

Discrete design space One simplification we have made throughout the whole thesis is that discrete design parameters are omitted and assumed to be given and fixed by a user. These parameters include a robot's frame topology or controller's architecture, and attempts to optimize them are common in real-world robot design tasks. As discrete parameters are inherently non-differentiable, our computational pipeline grounded on differentiable simulation does not apply. Therefore, we must introduce new techniques for at least two critical components in computational design with discrete parameters: First, a proper representation of the discrete design space. Second, and more importantly, an efficient algorithm for exploration and optimization in such a design space. A possible solution to both of them is to rewrite discrete designs as programs, e.g., using a concise program to describe the network architecture of a neural network controller. Following this idea, we can bridge computational design and program synthesis techniques to explore discrete design parameters [42]. Other possible solutions like representing discrete designs as graph networks are also viable, and we leave them as future work.

Multi-objective performance metrics So far in this thesis, we have only discussed designing a robot with a single performance metric. However, robots in real life are versatile, general-purpose, and rarely designed for maximizing a single objective only. Therefore, extending the entire computational design pipeline with multiple performance metrics would be an exciting direction. It turns out that optimizing with multiple objectives is generally more challenging than single-objective optimization because different objectives may correlate, conflict, or even compete with each other. In this case, one typically expects to explore a set of solutions with tradeoffs, known as the *Pareto* set, instead of finding a single solution that maximizes all objectives simultaneously. Introducing classical numerical methods in multi-objective optimization to computational robot design will inspire multi-purpose robot designs, which we consider a valuable future direction to explore.

Advanced physics simulation Another future direction we would like to explore is to create high-fidelity, multi-physics differentiable simulation. Despite the success in multicopters and soft underwater robots, the differentiable simulators presented in this thesis still have quite a few limitations in physics accuracy, collision handling, and fluid-solid coupling. Previous papers have provided solutions to simulating much more sophisticated physics systems than what we present in this thesis. In the future, we would like to revisit these forward simulation methods but from a perspective of differentiable simulation. As we have shown in Chpt. 4, augmenting an existing forward simulator with gradients can be nontrivial and lead to open research problems.

Sim-to-real transfer Lastly, although we have discussed the issue of sim-to-real gaps in this thesis, the solution we provide is not verified in more general cases other than the soft underwater robot we have shown in Chpt. 5. One strong assumption we have in Chpt. 5 is that the first-principle dynamic model has enough capacity to fit the motion data we gather from real-world experiments. However, a real-world robot model may contain uncertainties that are challenging to model with analytical governing equations. Moreover, even if it is possible to model the real-world dynamics with known equations, measuring their coefficients may be intractable in an experiment. One possible solution is to combine first-principle physics models with data-driven, learning-based approaches to absorb any uncertainties in the dynamics model, which has already drawn attention from researchers in the learning and robotics communities.

Apart from the directions mentioned above, we reiterate that computational robot design is a multidisciplinary problem requiring efforts from computer science, electric engineering, mechanical engineering, and material science, to name a few. The computational methodology presented in this thesis constitutes only a small portion of the optimal solution to this problem. Still, we believe our expedition in this thesis could be beneficial to the general audience interested in this problem, and we hope it could inspire future collaborations across different research fields including graphics, robotics, learning, programming language, and many more.

Bibliography

[1] Niels Aage, Erik Andreassen, Boyan S. Lazarov, and Ole Sigmund. Giga-voxel computational morphogenesis for structural design. *Nature*, 550(7674):84–86, 2017.

[2] Joe Alexandersen and Casper Schousboe Andreasen. A review of topology optimisation for fluid-based problems. *Fluids*, 5(1), 2020.

[3] Yara Almubarak, Matthew Punnoose, Nicole Xiu Maly, Armita Hamidi, and Yonas Tadesse. KryptoJelly: A jellyfish robot with confined, adjustable pre-stress, and easily replaceable shape memory alloy NiTi actuators. *Smart Materials and Structures*, 29(7):075011, jun 2020.

[4] Ryoichi Ando, Nils Thürey, and Chris Wojtan. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph.*, 32(4), July 2013.

[5] APM. APM autopilot suite. `http://ardupilot.org`, 2015.

[6] Karl Johan Aström and Richard M Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[7] Vinicius C. Azevedo, Christopher Batty, and Manuel M. Oliveira. Preserving geometry and topology for fluid flows with thin obstacles and narrow gaps. *ACM Trans. Graph.*, 35(4), July 2016.

[8] Moritz Bächer, Bernd Bickel, Emily Whiting, and Olga Sorkine-Hornung. Spin-it: Optimizing moment of inertia for spinnable objects. *Commun. ACM*, 60(8):92–99, July 2017.

[9] David L. Barrow and Philip W. Smith. Spline notation applied to a volume problem. *The American Mathematical Monthly*, 86(1):50–51, 1979.

[10] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, page 100–es, New York, NY, USA, 2007. Association for Computing Machinery.

[11] William Baxter, Yuanxin Liu, and Ming C. Lin. A viscous paint model for interactive applications. *Computer Animation and Virtual Worlds*, 15(3-4):433–441, 2004.

[12] Jacob Bedrossian, James H. von Brecht, Siwei Zhu, Eftychios Sifakis, and Joseph M. Teran. A second order virtual node method for elliptic problems with interfaces and irregular domains. *Journal of Computational Physics*, 229(18):6405–6426, 2010.

[13] Reza Behrou, Ram Ranjan, and James K. Guest. Adaptive topology optimization for incompressible laminar flow problems with mass flow constraints. *Computer Methods in Applied Mechanics and Engineering*, 346:612–641, 2019.

[14] Martin Philip Bendsoe and Ole Sigmund. *Topology Optimization: Theory, Methods, and Applications*. Springer Science & Business Media, 2013.

[15] James Bern, Pol Banzet, Roi Poranne, and Stelian Coros. Trajectory optimization for cable-driven soft robot locomotion. In *Proceedings of Robotics: Science and Systems*, Freiburg im Breisgau, Germany, June 2019.

[16] James M. Bern, Yannick Schnider, Pol Banzet, Nitish Kumar, and Stelian Coros. Soft robot control with a learned differentiable model. In *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, pages 417–423, 2020.

[17] Gaurav Bharaj, Stelian Coros, Bernhard Thomaszewski, James Tompkin, Bernd Bickel, and Hanspeter Pfister. Computational design of walking automata. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '15, page 93–100, New York, NY, USA, 2015. Association for Computing Machinery.

[18] Haimasree Bhatacharya, Michael B. Nielsen, and Robert Bridson. Steady state Stokes flow interpolation for fluid control. In Carlos Andujar and Enrico Puppo, editors, *Eurographics 2012 - Short Papers*. The Eurographics Association, 2012.

[19] Robert Bogue. Underwater robots: A review of technologies and applications. *Industrial Robot: An International Journal*, 42(3):186–191, Jan 2015.

[20] Josh Bongard, Cecilia Laschi, Hod Lipson, Nick Cheney, and Francesco Corucci. Material Properties Affect Evolutions Ability to Exploit Morphological Computation in Growing Soft-Bodied Creatures. volume ALIFE 2016, the Fifteenth International Conference on the Synthesis and Simulation of Living Systems of *ALIFE 2021: The 2021 Conference on Artificial Life*, pages 234–241, 07 2016.

[21] Thomas Borrvall and Joakim Petersson. Topology optimization of fluids in Stokes flow. *International Journal for Numerical Methods in Fluids*, 41(1):77–107, 2003.

[22] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4), July 2014.

[23] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[24] Christopher Brandt, Elmar Eisemann, and Klaus Hildebrandt. Hyper-reduced projective dynamics. *ACM Trans. Graph.*, 37(4), July 2018.

[25] Franco Brezzi and Michel Fortin. *Mixed and Hybrid Finite Element Methods*, volume 15. Springer Science & Business Media, 2012.

[26] Robert Bridson. *Fluid Simulation for Computer Graphics.* CRC press, 2015.

[27] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979, 2019.

[28] Desai Chen, David I. W. Levin, Wojciech Matusik, and Danny M. Kaufman. Dynamics-aware numerical coarsening for fabrication design. *ACM Trans. Graph.*, 36(4), July 2017.

[29] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. *SIGEVOlution*, 7(1):11–23, August 2014.

[30] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, New York, NY, USA, 2011. Association for Computing Machinery.

[31] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Trans. Graph.*, 32(4), July 2013.

[32] Francesco Corucci, Nick Cheney, Hod Lipson, Cecilia Laschi, and Josh Bongard. Evolving swimming soft-bodied creatures. In *ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, Late Breaking Proceedings*, volume 6, 2016.

[33] Keenan Crane. *Keenan's 3D Model Repository*, 2020.

[34] Christophe Daux, Nicolas Moës, John Dolbow, Natarajan Sukumar, and Ted Belytschko. Arbitrary branched and intersecting cracks with the extended finite element method. *International Journal for Numerical Methods in Engineering*, 48(12):1741–1760, 2000.

[35] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and

R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[36] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13:6, 2019.

[37] Raphael Deimel, Patrick Irmisch, Vincent Wall, and Oliver Brock. Automated co-design of soft hand morphology and control strategy for grasping. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1213–1218, 2017.

[38] Marc Deisenroth, Carl Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[39] Dimitar Dinev, Tiantian Liu, and Ladislav Kavan. Stabilizing integrators for real-time physics. *ACM Trans. Graph.*, 37(1), January 2018.

[40] Dimitar Dinev, Tiantian Liu, Jing Li, Bernhard Thomaszewski, and Ladislav Kavan. FEPR: Fast energy projection for real-time simulation of deformable objects. *ACM Trans. Graph.*, 37(4), July 2018.

[41] Tao Du, Josie Hughes, Sebastien Wah, Wojciech Matusik, and Daniela Rus. Underwater soft robot modeling and control with differentiable simulation. *IEEE Robotics and Automation Letters*, 6(3):4994–5001, 2021.

[42] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. InverseCSG: Automatic conversion of 3d models to CSG trees. *ACM Trans. Graph.*, 37(6), December 2018.

[43] Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. Computational multicopter design. *ACM Trans. Graph.*, 35(6), November 2016.

[44] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. DiffPD: Differentiable projective dynamics with contact. *ACM Trans. Graph. (accepted with minor revisions)*, 2021.

[45] Tao Du, Kui Wu, Andrew Spielberg, Wojciech Matusik, Bo Zhu, and Eftychios Sifakis. Functional optimization of fluidic devices with differentiable stokes flow. *ACM Trans. Graph.*, 39(6), November 2020.

[46] Hadi El Daou, Taavi Salumäe, Lily D Chambers, William M Megill, and Maarja Kruusmaa. Modelling of a biologically inspired robotic fish driven by compliant parts. *Bioinspiration & Biomimetics*, 9(1):016010, 2014.

[47] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *International Conference on Learning Representations*, 2019.

[48] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21(3):736–744, July 2002.

[49] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 15–22, New York, NY, USA, 2001. Association for Computing Machinery.

[50] Ronald P Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *Journal of Computational Physics*, 152(2):457–492, 1999.

[51] Bryan E. Feldman, James F. O'Brien, Bryan M. Klingner, and Tolga G. Goktekin. Fluids in deforming meshes. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, page 255–259, New York, NY, USA, 2005. Association for Computing Machinery.

[52] Florian Ferstl, Rüdiger Westermann, and Christian Dick. Large-scale liquid simulation on adaptive hexahedral grids. *IEEE Transactions on Visualization and Computer Graphics*, 20(10):1405–1417, 2014.

[53] Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. Vivace: A practical Gauss-Seidel method for stable soft body dynamics. *ACM Trans. Graph.*, 35(6), November 2016.

[54] Akash Garg, Andrew O. Sageman-Furnas, Bailin Deng, Yonghao Yue, Eitan Grinspun, Mark Pauly, and Max Wardetzky. Wire mesh design. *ACM Trans. Graph.*, 33(4), July 2014.

[55] Francisco. J. Gaspar, José L. Gracia, Francisco J. Lisbona, and Cornelis W. Oosterlee. Distributive smoothers in multigrid for problems with dominating grad–div operators. *Numerical Linear Algebra with Applications*, 15(8):661–683, 2008.

[56] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. ADD: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Trans. Graph.*, 39(6), November 2020.

[57] A. N. Gent. On the relation between indentation hardness and Young's modulus. *Rubber Chemistry and Technology*, 31(4):896–906, 09 1958.

[58] Allan Gersborg-Hansen, Ole Sigmund, and Robert B. Haber. Topology optimization of channel flow problems. *Structural and Multidisciplinary Optimization*, 30(3):181–192, Sep 2005.

[59] Michael Grant and Stephen Boyd. CVX: MATLAB software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, 2014.

[60] Michael C. Grant and Stephen P. Boyd. Graph implementations for nonsmooth convex programs. In Vincent D. Blondel, Stephen P. Boyd, and Hidenori Kimura, editors, *Recent Advances in Learning and Control*, pages 95–110, London, 2008. Springer London.

[61] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[62] David Hahn, Pol Banzet, James M. Bern, and Stelian Coros. Real2Sim: Viscoelastic parameter estimation from dynamic motion. *ACM Trans. Graph.*, 38(6), November 2019.

[63] Xuchen Han, Theodore F. Gast, Qi Guo, Stephanie Wang, Chenfanfu Jiang, and Joseph Teran. A hybrid material point method for frictional contact with diverse materials. *Proc. ACM Comput. Graph. Interact. Tech.*, 2(2), July 2019.

[64] Jeffrey Lee Hellrung, Luming Wang, Eftychios Sifakis, and Joseph M. Teran. A second order virtual node method for elliptic problems with interfaces and irregular domains in three dimensions. *J. Comput. Phys.*, 231(4):2015–2048, February 2012.

[65] Philipp Herholz and Olga Sorkine-Hornung. Sparse Cholesky updates for interactive mesh parameterization. *ACM Trans. Graph.*, 39(6), November 2020.

[66] Jonathan Hiller and Hod Lipson. Dynamic simulation of soft multimaterial 3D-printed objects. *Soft Robotics*, 1(1):88–101, 2014.

[67] Philipp Holl, Nils Thuerey, and Vladlen Koltun. Learning to control PDEs with differentiable physics. In *International Conference on Learning Representations*, 2020.

[68] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations*, 2020.

[69] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B. Tenenbaum, William T. Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. ChainQueen: A real-time differentiable physical simulator for soft robotics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6265–6271, 2019.

[70] Weicheng Huang, Zachary Patterson, Carmel Majidi, and M Khalid Jawed. Modeling soft swimming robots using discrete elastic rod method. In *Bioinspired Sensing, Actuation, and Control in Underwater Soft Robotic Systems*, pages 247–259. Springer, 2021.

[71] Thomas J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis.* Courier Corporation, 2012.

[72] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

[73] Hikaru Ibayashi, Chris Wojtan, Nils Thuerey, Takeo Igarashi, and Ryoichi Ando. Simulating liquids on dynamically warping grids. *IEEE Transactions on Visualization and Computer Graphics*, 26(6):2288–2302, 2020.

[74] Sumit Jain and C. Karen Liu. Interactive synthesis of human-object interaction. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, page 47–53, New York, NY, USA, 2009. Association for Computing Machinery.

[75] Doug L. James and Dinesh K. Pai. ArtDefo: Accurate real time deformable objects. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, page 65–72, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[76] Eunjung Ju, Jungdam Won, Jehee Lee, Byungkuk Choi, Junyong Noh, and Min Gyu Choi. Data-driven control of flapping flight. *ACM Trans. Graph.*, 32(5), October 2013.

[77] Robert K. Katzschmann, Joseph DelPreto, Robert MacCurdy, and Daniela Rus. Exploration of underwater life with an acoustically controlled soft robotic fish. *Science Robotics*, 3(16), 2018.

[78] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[79] Bongjin Koo, Wilmot Li, JiaXian Yao, Maneesh Agrawala, and Niloy J. Mitra. Creating works-like prototypes of mechanical objects. *ACM Trans. Graph.*, 33(6), November 2014.

[80] Vladislav Kopman, Jeffrey Laut, Francesco Acquaviva, Alessandro Rizzo, and Maurizio Porfiri. Dynamic modeling of a robotic fish propelled by a compliant tail. *IEEE Journal of Oceanic Engineering*, 40(1):209–221, 2015.

[81] Nipun Kwatra, Jonathan Su, Jón T. Grétarsson, and Ronald Fedkiw. A method for avoiding the acoustic time step restriction in compressible flow. *Journal of Computational Physics*, 228(11):4146–4161, 2009.

[82] Egor Larionov, Christopher Batty, and Robert Bridson. Variational Stokes: A unified pressure-viscosity solver for accurate viscous liquids. *ACM Trans. Graph.*, 36(4), July 2017.

[83] Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. Converting 3D furniture models to fabricatable parts and connectors. *ACM Trans. Graph.*, 30(4), July 2011.

[84] Alan Laub. A Schur method for solving algebraic Riccati equations. *IEEE Transactions on Automatic Control*, 24(6):913–921, 1979.

[85] Benny Lautrup. *Physics of Continuous Matter: Exotic and Everyday Phenomena in the Macroscopic World.* CRC press, 2004.

[86] Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Trans. Graph.*, 28(4), September 2009.

[87] J Gordon Leishman. *Principles of Helicopter Aerodynamics.* Cambridge University Press, 2006.

[88] Michael Lentine, Jón Tómas Grétarsson, Craig Schroeder, Avi Robinson-Mosher, and Ronald Fedkiw. Creature control in a fluid environment. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):682–693, 2011.

[89] Jing Li, Tiantian Liu, and Ladislav Kavan. Fast simulation of deformable characters with articulated skeletons in projective dynamics. In *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '19, New York, NY, USA, 2019. Association for Computing Machinery.

[90] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. Incremental potential contact: Intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 39(4), July 2020.

[91] Tiefeng Li, Guorui Li, Yiming Liang, Tingyu Cheng, Jing Dai, Xuxu Yang, Bangyuan Liu, Zedong Zeng, Zhilong Huang, Yingwu Luo, Tao Xie, and Wei Yang. Fast-moving soft electronic fish. *Science Advances*, 3(4), 2017.

[92] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2019.

[93] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[94] Sen Lin, Longyu Zhao, James K. Guest, Timothy P. Weihs, and Zhenyu Liu. Topology optimization of fixed-geometry fluid diodes. *Journal of Mechanical Design*, 137(8), 08 2015. 081402.

[95] Haixiang Liu, Yuanming Hu, Bo Zhu, Wojciech Matusik, and Eftychios Sifakis. Narrow-band topology optimization on a sparsely populated grid. *ACM Trans. Graph.*, 37(6), December 2018.

[96] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Quasi-Newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph.*, 36(4), May 2017.

[97] Mickaël Ly, Jean Jouve, Laurence Boissieux, and Florence Bertails-Descoubes. Projective dynamics with dry frictional contact. *ACM Trans. Graph.*, 39(4), July 2020.

[98] Pingchuan Ma, Tao Du, and Wojciech Matusik. Efficient continuous pareto exploration in multi-task learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6522–6531. PMLR, 13–18 Jul 2020.

[99] Pingchuan Ma, Tao Du, John Z. Zhang, Kui Wu, Andrew Spielberg, Robert K. Katzschmann, and Wojciech Matusik. DiffAqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation. *ACM Trans. Graph.*, 40(4), July 2021.

[100] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and T. Y. Kim. *Primal/Dual Descent Methods for Dynamics*. Eurographics Association, Goslar, DEU, 2020.

[101] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, New York, NY, USA, 2011. Association for Computing Machinery.

[102] Tobias Martin, Nobuyuki Umetani, and Bernd Bickel. OmniAD: Data-driven omni-directional aerodynamics. *ACM Trans. Graph.*, 34(4), July 2015.

[103] MATLAB. Linear-Quadratic Regulator (LQR) design. `http://www.mathworks.com/help/control/ref/lqr.html`, 2016.

[104] Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.*, 30(4), July 2011.

[105] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3):449–456, August 2004.

[106] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1):21–39, 2012.

[107] Sehee Min, Jungdam Won, Seunghwan Lee, Jungnam Park, and Jehee Lee. SoftCon: Simulation and control of soft-bodied animals with biomimetic actuators. *ACM Trans. Graph.*, 38(6), November 2019.

[108] Nicolas Moës, John Dolbow, and Ted Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(1):131–150, 1999.

[109] Yuki Mori and Takeo Igarashi. Plushie: An interactive design system for plush toys. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, page 45–es, New York, NY, USA, 2007. Association for Computing Machinery.

[110] Jorge Nocedal and Stephen Wright. *Numerical Optimization.* Springer Science & Business Media, 2006.

[111] Maxim Olshanskii, Gert Lube, Timo Heister, and Johannes Löwe. Grad–div stabilization and subgrid pressure models for the incompressible Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 198(49):3975–3988, 2009.

[112] Stanley Osher and Ronald Fedkiw. Level set methods and dynamic implicit surfaces. *Surfaces*, 44(77):685, 2002.

[113] Matthew Overby, George E. Brown, Jie Li, and Rahul Narain. ADMM $\supseteq$ projective dynamics: Fast simulation of hyperelastic models with dynamic constraints. *IEEE Transactions on Visualization and Computer Graphics*, 23(10):2222–2234, 2017.

[114] D. Ann Pabst. Springs in swimming animals. *American Zoologist*, 36(6):723–735, 08 2015.

[115] Taylor Patterson, Nathan Mitchell, and Eftychios Sifakis. Simulation of complex nonlinear elastic bodies using lattice deformers. *ACM Trans. Graph.*, 31(6), November 2012.

[116] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810, 2018.

[117] Jovan Popović, Steven M. Seitz, and Michael Erdmann. Motion sketching for control of rigid-body simulations. *ACM Trans. Graph.*, 22(4):1034–1054, October 2003.

[118] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 209–217, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[119] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Scalable differentiable physics for learning and control. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7847–7856. PMLR, 13–18 Jul 2020.

[120] Federico Renda, Francesco Giorgio-Serchi, Frédéric Boyer, and Cecilia Laschi. Modelling cephalopod-inspired pulsed-jet locomotion for underwater soft robots. *Bioinspiration & Biomimetics*, 10(5):055005, 2015.

[121] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8459–8468. PMLR, 13–18 Jul 2020.

[122] Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. SketchChair: An all-in-one chair design system for end users. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '11, page 73–80, New York, NY, USA, 2010. Association for Computing Machinery.

[123] Connor Schenck and Dieter Fox. SPNets: Differentiable fluid dynamics for deep neural networks. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 317–335. PMLR, 29–31 Oct 2018.

[124] Olaf Schenk and Klaus Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Generation Computer Systems*, 20(3):475–487, 2004.

[125] Craig Schroeder, Wen Zheng, and Ronald Fedkiw. Semi-implicit surface tension formulation with a Lagrangian surface mesh on an Eulerian simulation grid. *J. Comput. Phys.*, 231(4):2092–2115, February 2012.

[126] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[127] Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. Design and fabrication by example. *ACM Trans. Graph.*, 33(4), July 2014.

[128] SciPy. CARE solver. `http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.linalg.solve_continuous_are.html`, 2014.

[129] Jun Shintake, Vito Cacucciolo, Herbert Shea, and Dario Floreano. Soft biomimetic fish robot made of dielectric elastomer actuators. *Soft Robotics*, 5(4):466–474, 2018. PMID: 29957131.

[130] Eftychios Sifakis and Jernej Barbic. FEM simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, New York, NY, USA, 2012. Association for Computing Machinery.

[131] Eftychios Sifakis and Jernej Barbic. FEM simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, New York, NY, USA, 2012. Association for Computing Machinery.

[132] Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. Hybrid simulation of deformable solids. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, page 81–90, Goslar, DEU, 2007. Eurographics Association.

[133] Mélina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. Designing inflatable structures. *ACM Trans. Graph.*, 33(4), July 2014.

[134] Inc. Smooth-On. Soma Foama 25 product specification. `https://www.smooth-on.com/tb/files/SOMA_FOAMA_TB.pdf`. Accessed: 2021-02-07.

[135] Luca Somm, David Hahn, Nitish Kumar, and Stelian Coros. Expanding foam as the material for fabrication, prototyping and experimental assessment of low-cost soft robots with embedded sensing. *IEEE Robotics and Automation Letters*, 4(2):761–768, 2019.

[136] Andrew Spielberg, Allan Zhao, Yuanming Hu, Tao Du, Wojciech Matusik, and Daniela Rus. Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[137] Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, page 121–128, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[138] Alexey Stomakhin, Craig Schroeder, Chenfanfu Jiang, Lawrence Chai, Joseph Teran, and Andrew Selle. Augmented MPM for phase-change and varied materials. *ACM Trans. Graph.*, 33(4), July 2014.

[139] Andrew Stuart and Tony Humphries. Dynamical systems and numerical analysis. *Cambridge University*, 1996.

[140] Jie Tan, Yuting Gu, C. Karen Liu, and Greg Turk. Learning bicycle stunts. *ACM Trans. Graph.*, 33(4), July 2014.

[141] Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. Articulated swimming creatures. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, New York, NY, USA, 2011. Association for Computing Machinery.

[142] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

[143] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[144] Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. Keyframe control of smoke simulations. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, page 716–723, New York, NY, USA, 2003. Association for Computing Machinery.

[145] Christopher D. Twigg and Doug L. James. Backward steps in rigid body simulation. *ACM Trans. Graph.*, 27(3):1–10, August 2008.

[146] Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.*, 31(4), July 2012.

[147] Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. Sensitive couture for interactive garment modeling and editing. *ACM Trans. Graph.*, 30(4), July 2011.

[148] Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Trans. Graph.*, 33(4), July 2014.

[149] Alex Villanueva, Colin Smith, and Shashank Priya. A biomimetic robotic jellyfish (robojelly) actuated by shape memory alloy composite actuators. *Bioinspiration & biomimetics*, 6(3):036004, 2011.

[150] Carlos H. Villanueva and Kurt Maute. CutFEM topology optimization of 3D laminar incompressible flow problems. *Computer Methods in Applied Mechanics and Engineering*, 320:444–473, 2017.

[151] Etienne Vouga, Mathias Höbinger, Johannes Wallner, and Helmut Pottmann. Design of self-supporting surfaces. *ACM Trans. Graph.*, 31(4), July 2012.

[152] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, New York, NY, USA, 2009. Association for Computing Machinery.

[153] Huamin Wang. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph.*, 34(6), October 2015.

[154] Huamin Wang and Yin Yang. Descent methods for elastic body simulation on the GPU. *ACM Trans. Graph.*, 35(6), November 2016.

[155] Jianxun Wang, Philip K. McKinley, and Xiaobo Tan. Dynamic modeling of robotic fish with a base-actuated flexible tail. *Journal of Dynamic Systems, Measurement, and Control*, 137(1), 08 2014. 011004.

[156] Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Automatic robot design. In *International Conference on Learning Representations*, 2019.

[157] Zhenlong Wang, Guanrong Hang, Jian Li, Yangwei Wang, and Kai Xiao. A micro-robot fish with embedded SMA wire actuated flexible biomimetic fin. *Sensors and Actuators A: Physical*, 144(2):354–360, 2008.

[158] Emily Whiting, Hijung Shin, Robert Wang, John Ochsendorf, and Frédo Durand. Structural optimization of 3d masonry buildings. *ACM Trans. Graph.*, 31(6), November 2012.

[159] Chris Wojtan, Peter J. Mucha, and Greg Turk. Keyframe control of complex particle systems using the adjoint method. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '06, page 15–23, Goslar, DEU, 2006. Eurographics Association.

[160] Jia-chi Wu and Zoran Popović. Realistic modeling of bird flight animations. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, page 888–895, New York, NY, USA, 2003. Association for Computing Machinery.

[161] Zangyueyang Xian, Xin Tong, and Tiantian Liu. A scalable Galerkin multigrid method for real-time simulation of deformable objects. *ACM Trans. Graph.*, 38(6), November 2019.

[162] Jie Xu, Tao Du, Michael Foshey, Beichen Li, Bo Zhu, Adriana Schulz, and Wojciech Matusik. Learning to fly: Computational controller design for hybrid UAVs with reinforcement learning. *ACM Trans. Graph.*, 38(4), July 2019.

[163] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. TossingBot: Learning to throw arbitrary objects with residual physics. 2019.

[164] Yong Zhong, Zheng Li, and Ruxu Du. A novel robot fish with wire-driven active body and compliant tail. *IEEE/ASME Transactions on Mechatronics*, 22(4):1633–1643, 2017.

[165] Mingdong Zhou, Haojie Lian, Ole Sigmund, and Niels Aage. Shape morphing and topology optimization of fluid channels by explicit boundary tracking. *International Journal for Numerical Methods in Fluids*, 88(6):296–313, 2018.

[166] Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. Motion-guided mechanical toy modeling. *ACM Trans. Graph.*, 31(6), November 2012.

[167] Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph.*, 29(2), April 2010.

[168] Yongning Zhu, Yuting Wang, Jeffrey Hellrung, Alejandro Cantarero, Eftychios Sifakis, and Joseph M. Teran. A second-order virtual node algorithm for nearly incompressible linear elasticity in irregular domains. *J. Comput. Phys.*, 231(21):7092–7117, August 2012.