

ABSTRACT

Title of Dissertation: **MACHINE LEARNING WITH
DIFFERENTIABLE PHYSICS PRIORS**

Yiling Qiao
Doctor of Philosophy, 2024

Dissertation Directed by: **Professor Ming C. Lin**
Department of Computer Science

Differentiable physics priors enable gradient-based learning systems to adhere to physical dynamics. By making physics simulations differentiable, we can backpropagate through the physical consequences of actions. This pipeline allows agents to quickly learn to achieve desired effects in the physical world and is an effective technique for solving inverse problems in physical or dynamical systems. This new programming paradigm bridges model-based and data-driven methods, mitigating data scarcity and model bias simultaneously.

My research focuses on developing scalable, powerful, and efficient differentiable physics simulators. We have created state-of-the-art differentiable physics for rigid bodies, cloth, fluids, articulated bodies, and deformable solids, achieving performance orders of magnitude better than existing alternatives. These differentiable simulators are applied to solve inverse problems, train control policies, and enhance reinforcement learning algorithms.

MACHINE LEARNING WITH DIFFERENTIABLE PHYSICS PRIORS

by

Yiling Qiao

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:

Professor Ming C. Lin, Chair/Advisor
Professor Maria Cameron, Dean's Representative
Professor Jia-Bin Huang
Professor Tianyi Zhou
Professor Dinesh Manocha

© Copyright by
Yiling Qiao
2024

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Ming Lin. Your guidance, support, and encouragement have been invaluable throughout my doctoral journey. Your expertise and insights have profoundly shaped this dissertation, and I am incredibly fortunate to have had the opportunity to work under your mentorship. Your stories have inspired me greatly, and your suggestions about my future career will always play an important role in my life.

I am also profoundly grateful to the members of my dissertation committee, Dr. Maria Cameron, Dinesh Manocha, Jia-Bin Huang, and Tianyi Zhou, for their insightful feedback, constructive criticism, and unwavering support. Your contributions have significantly enhanced the quality of this work.

I would like to extend my heartfelt thanks to Dr. Vladlen Koltun, Aniket Bera, Matthias Zwicker, Tom Goldstein, Daniele Panozzo, Hao Su, Bo Zhu, Chuang Gan, Animesh Garg, Xiaodi Wu, Lingjie Liu, Breannan Smith, Takaaki Shiratori, Ken Museth, Miles Macklin, Chris Atkeson, Lin Gao, Xilin Chen, and others for your kind guidance and help. I feel so fortunate to have had the opportunity to read your great research works first and then to have been helped and/or mentored by you later.

To my friends and labmates in GAMMA, Aaron, Adarsh, Alexander, Bhrij, Divya, Geonsun, Jaehoon, Jing, Junbang, Justin, Laura, Mohamed, Niall, Peng, Pooja, Qingyang, Rohan, Ruicheng,

Ruiqi, Sanghyun, Senthil, Shrey, Shrellekha, Tianrui, Trisha, Utsav, Uttaran, Vishnu, Will, Xijun, Xiyang, Yonghan, Yu, Zhenyu, Jason, Tetsuya, Xian, Zhenjia, Johnson, Pingchuan, Xuan, Peter, Krishna, and others, thank you for your companionship, encouragement, and the countless discussions that have helped shape my thinking.

At the University of Maryland, I really spent five happy years. Thank you Tom, Jodie, Sharron, Migo, Shanshan, and all the other staff in ISSS, the Graduate School, the CS department, the Eppley Center, and elsewhere.

I also sincerely appreciate the support from Intel, Nvidia, and Meta for my research. A special thanks to the Meta Fellowship. Thank you for recognizing our research. That really means a lot to me and gives me the confidence for more ambitious ventures in the future.

A special note of appreciation goes to my family: my wife, mom, dad, grandma, uncle, and aunts. Thank you for your love, patience, and belief in me. To my housemates, Jiaqi, Yuxiang, Hsien-Yu, and Suying, thanks for all the good times we had together. All the best to your future journey as well.

Thank you all for your contributions, support, and encouragement. This dissertation would not have been possible without you.

Table of Contents

Acknowledgements	ii
Table of Contents	iv
List of Tables	viii
List of Figures	ix
I Introduction	1
Chapter 1: Overview	2
1.1 Motivation	2
1.2 Related work	3
1.2.1 Differentiable Programming	3
1.2.2 Simulation and Animation Techniques	6
1.2.3 Differentiable 3D Reconstruction	7
1.3 Outline of Dissertation	8
II Differentiable Simulation	12
Chapter 2: Scalable Differentiable Physics for Learning and Control	13
2.1 Introduction	13
2.2 Related Work	15
2.3 Overview	17
2.4 Preliminaries	18
2.5 Scalable Collision Handling	19
2.6 Fast Differentiation	22
2.7 Experiments	25
2.7.1 Scalability	26
2.7.2 Ablation Studies	27
2.7.3 Two-way coupling	30
2.7.4 Applications	30
2.8 Conclusion	36
Chapter 3: Efficient Differentiable Simulation of Articulated Bodies	38

3.1	Introduction	38
3.2	Related Work	40
3.3	Preliminaries	43
3.4	Efficient Differentiation	45
3.4.1	Adjoint Method for Articulated Dynamics	45
3.4.2	Checkpointing for Articulated Dynamics	48
3.5	Reinforcement Learning	50
3.6	Experiments	52
3.6.1	Comparison with Autodiff Tools	53
3.6.2	Integration with Reinforcement Learning	56
3.6.3	Applications	59
3.7	Conclusion	60
Chapter 4:	Differentiable Simulation of Soft Multi-body Systems	64
4.1	Related Work	66
4.2	Soft Body Simulation Using Projective Dynamics	68
4.3	Articulated Skeletons	70
4.3.1	Rigid Body System	70
4.3.2	Top-down Matrix Assembly for Articulated Body Systems	72
4.3.3	Articulated Joint Actuation	74
4.4	Contact Modeling	75
4.5	Experiments	77
4.5.1	Implementation	77
4.5.2	Ablation Study	79
4.5.3	Applications	80
4.6	Conclusion	83
Chapter 5:	Other Differentiable Dynamical Systems	85
5.1	Traffic Simulation	85
5.2	Fluids Simulation	87
5.3	Analog Quantum Computing	90

III Reality-based Simulation 95

Chapter 6:	Editable Neural Geometry and Physics from Monocular Videos	96
6.1	Introduction	96
6.2	Related Work	99
6.3	Methods	100
6.3.1	Rendering Pipeline	100
6.3.2	Simulation Pipeline	102
6.3.3	Training and Losses	107
6.4	Experiments	109
6.4.1	Dynamic Geometry Reconstruction	109
6.4.2	Novel View Synthesis	109

6.4.3	Parameter Estimation	111
6.4.4	Editing	111
6.4.5	Ablation Study	113
6.5	Conclusion	117
Chapter 7: Dynamic Mesh-Aware Radiance Fields		118
7.1	Introduction	118
7.2	Related Work	120
7.2.1	Neural Fields and Surface Representations	120
7.2.2	Scene Editing and Synthesis	122
7.3	Method	123
7.3.1	Rendering	123
7.3.2	Simulation	131
7.3.3	Implementation Details	133
7.4	Experiments	133
7.4.1	Comparisons	133
7.4.2	Performance	137
7.4.3	Applications	137
7.5	Conclusion	138
Chapter 8: Physically Consistent Perception of Hand-Object Interactions with Differentiable Priors		141
8.1	Introduction	141
8.2	Related Works	144
8.3	Methods	144
8.3.1	Problem Setting	146
8.3.2	Differentiable Rendering Priors	147
8.3.3	Differentiable Physics Priors	148
8.3.4	Optimization-based Refinement	149
8.3.5	Filtering-based Tracking	150
8.4	Experiments	152
8.4.1	Real-world Robotic Hand and Object Iteration	153
8.4.2	Pose Estimation	155
8.4.3	Filtering-based tracking	157
8.4.4	Contact Refinement	158
8.4.5	Generalization to Human	159
8.5	Conclusion	160
Chapter 9: Other Visual-Dynamical Systems		162
9.1	Navigation and Collision Avoidance Using Commodity Sensors	162
9.2	Physics Augmented Continuum Neural Radiance Fields	164

IV Conclusion	167
Chapter 10: Conclusion	168
10.1 Summary of Results	168
10.2 Future Work	170
 V Appendix	 173
Appendix A: Scalable Differentiable Physics for Learning and Control	174
Appendix B: Efficient Differentiable Simulation of Articulated Bodies	180
Appendix C: Differentiable Simulation of Soft Multi-body Systems	201
Appendix D: Editable Neural Geometry and Physics from Monocular Videos	209
Appendix E: Dynamic Mesh-Aware Radiance Fields	213
Appendix F: Physically Consistent Perception of Hand-Object Interactions with Differentiable Priors	223

List of Tables

2.1	Performance of collision handling	29
2.2	Performance of differentiation	29
3.1	Memory usage of different simulation pipelines	52
3.2	Simulation time for forward pass	53
3.3	Simulation time for the backward pass	55
4.1	Memory footprint (GB).	78
5.1	Machine learning in different dynamical systems.	91
6.1	Quantitative evaluation of video reconstruction	109
6.2	Quantitative evaluation of novel view synthesis	110
8.1	Ablation study	150
8.2	Evaluation of hand pose estimation	154
8.3	Quantitative evaluation of object pose estimation	155
8.4	Error after contact-based pose refinement	159
8.5	Quantitative evaluation on human object pose estimation	159
B.1	Maximum reward on n -link pendulum	182
B.2	Model transplant results on MuJoCo Ant	185
F.1	Error after contact-based pose refinement	225

List of Figures

1.1	An overview of my research. Upper boxes are the closed-loop pipeline, and the lower boxes are my works.	3
2.1	Differentiable simulation embedded in a neural network	17
2.2	Collision visualization	22
2.3	Scalability comparison with grid-based methods	25
2.4	Ablation studies for collision handling	28
2.5	Two-way coupling of rigid bodies and cloth	31
2.6	Comparison with MuJoCo	32
2.7	Comparison on inverse problems	33
2.8	Learning control policy	34
2.9	Parameter estimation	35
2.10	Interoperability across simulators	36
3.1	The workflow of a simulation step	43
3.2	Differentiation of articulated body dynamics	49
3.3	Memory and speed comparison	54
3.4	The n -link pendulum task	57
3.5	The MuJoCo Ant task	59
3.6	Motion control task	62
3.7	Parameter estimation	63
4.1	Ablation study of skeleton realization	78
4.2	Ablation study of collision handling scheme	80
4.3	System identification	81
4.4	Motion control experiments	83
5.1	Traffic simulation in an urban environment	86
6.1	Overall workflow of NeuPhysics	101
6.2	The editing pipeline	104
6.3	Computational Graph During Training	107
6.4	Surface reconstruction from dynamic videos	109
6.5	Qualitative evaluation of geometry reconstruction and novel view synthesis	110
6.6	Parameter estimation and editing by differentiable physics	110

6.7	Geometry and appearance editing in videos	112
6.8	Joint training can be time-consuming and converges slowly.	113
6.9	Region of Interest	115
6.10	Compare with optimizing Chamfer Distance	117
7.1	Light transport on the surface (<i>left</i>) and in the medium (<i>right</i>).	119
7.2	Pipeline overview	121
7.3	Shadow Casting	123
7.4	HDR Volumetric Radiance Map	128
7.5	Rendering comparison for virtual object insertion	132
7.6	Qualitative results for simulation	134
7.7	Real-time photorealistic gaming on a laptop	136
7.8	Application of mixing NeRF with meshes	139
8.1	Application in multiple tasks	142
8.2	An overview of our optimization and filtering pipelines	144
8.3	Diagram of the EKF filtering process	150
8.4	Real-world pose estimation	153
8.5	Pose Estimation Results	154
8.6	Filtering results	157
8.7	Qualitative results of contact-based pose refinement	158
8.8	Human Object Pose Refinement	160
A.1	Qualitative comparison on complex models and interactions	178
B.1	Ablation study for the checkpointing scheme.	183
C.1	Ablation study of embedded skeletons	207
C.2	Ablation study of collision handling scheme	207
D.1	Qualitative evaluation of video reconstruction	211
E.1	Optimizing Emission UV Texture Map	217
E.2	Inverse Camera Response Function (CRF)	219
E.3	Scaling the number of the inserted objects and image resolution	220
E.4	Run time comparison with NeuPhysics	220
E.5	Simulation Comparison	221
E.6	Number of Rank-1s each method receives.	221
E.7	Comparison between Luma AI (<i>left</i>) and Ours (<i>right</i>).	222

Part I

Introduction

Chapter 1: Overview

1.1 Motivation

Physically-based simulation has become an essential component in animation, visual effects, and broader computational problems, such as aircraft design and digital fabrication. In the recent wave of machine learning, simulation environments for games, robots, and autonomous cars have played a crucial role as infrastructure, fostering breakthroughs in learning algorithms. With stronger artificial general intelligence approaching, I believe that simulation will be even more important. We need physics-informed AI; any intelligent agent needs to be safely trained, aligned, and verified in a simulated sandbox before it can ultimately be deployed and benefit our human society. Moreover, despite the presence of numerous simulation models and real-world data, people are working to address the sim-to-real (generalization and robustness problems) and real-to-sim (digital twin) gaps. Simply enlarging the dataset size or pursuing infinitely accurate simulation would be too expensive. Instead, I aim to leverage the merits of both data-driven and model-based approaches by designing novel physics-informed learning algorithms. In recent years, I have undertaken many pioneering works in **differentiable simulation** toward this goal. My research goal is to develop safe, realistic, and efficient simulations for graphics, robotics, embodied AI, digital twin, and inverse design.

The core of my work can be distilled into a three-part closed-loop pipeline, as illustrated

in Figure 1. (1) Formulate the governing dynamics: I will initially define the underlying rules of the system and tailor the simulation to be efficient and realistic for machine learning algorithms. (2) Generate and understand observations: Subsequently, coupling the simulation output with real-world observations, the existing data can be used to refine our model-based approach by automatically finetuning the simulation parameters. (3) Interact and control the system: With prior knowledge of the model and insights gained from real data, our algorithms can make more informed decisions to interact with the system. Throughout this closed-loop pipeline, my collaborators and I have developed several software libraries and made significant contributions published in internationally renowned venues, as ICML, NeurIPS, ICLR, Siggraph Asia, ICRA, and ICCV. Those works have garnered significant interest from researchers across the Machine Learning, Computer Graphics, Robotics, and Computer Vision research communities.

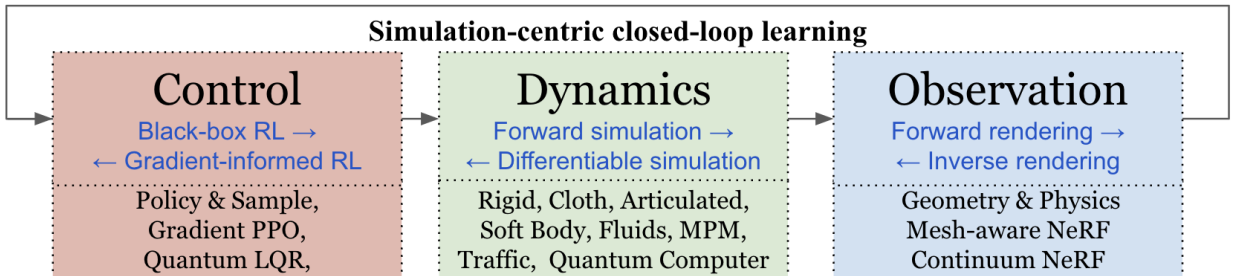


Figure 1.1: An overview of my research. Upper boxes are the closed-loop pipeline, and the lower boxes are my works.

1.2 Related work

1.2.1 Differentiable Programming

Differentiable physics provides gradients for learning, control, and inverse problems that involve physical systems. [1] advocate using differentiable physics to solve control problems in robotics. [2] obtain gradients of 2D rigid body dynamics. [3] use automatic differentiation

tools to obtain gradients for cloth simulation. [4] develop a more comprehensive differentiable physics engine for rigid bodies and cloth based on mesh representations. [5] present differentiable simulation of protein dynamics. For volumetric data, ChainQueen [6] computes gradients of the MPM simulation pipeline. [7] use FEM to model soft robots and perform trajectory optimization with analytic gradients. [8] differentiate the simulation of fluids with solid coupling.

Many exciting applications of differentiable physics have been explored [9, 10, 11, 12]. [13] propose a soft-body manipulation benchmark for differentiable physics. [14] manipulate tools with the help of differentiable physics. [15] perform system identification by learning from the trajectory, and [16] then use the estimated frictional properties to help robotic pushing. Differentiable simulation can also be used to train control policies by optimizing model predictive control’s objective function [17, 18].

A related line of work concerns the development of powerful automatic differentiation tools that can be used to differentiate simulation. DiffTaichi [19] provides a new programming language and a compiler, enabling the high-performance Taichi simulator to compute the gradients of the simulation. JAX MD [20] makes use of the JAX autodiff library [21] to differentiate molecular dynamics simulation. These works have specifically made use of vectorization on both CPU and GPU to achieve high performance on grids and particle sets, where the simulation is intrinsically parallel. In contrast, the simulation of articulated bodies is far more serial, as dynamics propagates along kinematic paths rather than acting in parallel on grid cells or particles.

TinyDiffSim [22] provides a templated simulation framework that can leverage existing autodiff tools such as CppAD [23], Ceres [24], and PyTorch [25] to differentiate simulation. However, these methods introduce significant overhead in tracing the computation graph and accumulate substantial computational and memory costs when applied to articulated bodies.

Our method does not need to store the entire computation graph to compute the gradients. We store the initial states of each simulation step and reproduce the intermediate results when needed by the backward pass. This checkpointing strategy is also used in training neural ODEs [26] and large neural networks [27, 28, 29, 30]. We are the first to adapt this technique to articulated dynamics, achieving dramatic reductions in memory consumption and enabling stable simulation of long experiences.

Adjoint method. The adjoint method has been applied to fluid control [31], PDEs [32], light transport simulation [33], and neural ODEs [26, 34]. Recently, [35] proposed to use the adjoint method in multi-body dynamics. However, they operate in maximal coordinates and model body attachments using springs. This does not enforce physical validity of articulated bodies. In contrast, we operate in reduced coordinates and derive the adjoints for the articulated body algorithm and spatial algebra operators that properly model the body’s joints. This supports physically correct simulation with joint torques, limits, Coriolis forces, and proper transmission of internal forces between links.

Neural approximation. A number of works approximate physics simulation using neural networks [36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46]. Physical principles have also been incorporated in the design of neural networks [47, 48, 49, 50, 51]. Approximate simulation by neural networks is naturally differentiable, but the networks are not constrained to abide by the underlying physical dynamics and simulation fidelity may degenerate outside the training distribution.

1.2.2 Simulation and Animation Techniques

Deformable body simulation using Finite Element Method (FEM) plays an important role in many scientific and engineering problems [52, 53, 54]. Previous works model soft bodies using different representations and methods for specific tasks. There are several kinds of approaches for modeling body actuation. Pneumatic-based methods [55] change the rest shape to produce reaction forces. Rigid bones attached within soft materials are also used to control the motion of deformable bodies [56, 57, 58, 59]. To further simulate biologically realistic motion, it is common to apply joint torques in articulated skeletons [60]. For example, [61, 62] use articulated body dynamics to govern the motion while handling collisions using soft contact. Inspired by animals, different designs of muscle-like actuators for soft-body simulations were also proposed [63, 64, 65].

Regarding contact modeling, spring-based penalty forces are widely used [66, 67, 68] for their simplicity. More advanced algorithms include inelastic projection [69, 70] and barrier-based repulsion [71, 72]. However, these methods do not always conform to Coulomb’s frictional law. We opt for a more realistic dry frictional model [73] to better handle collisions.

Projective Dynamics [74] is widely used for its robustness and efficiency for implicit time integration. It has been extended to model muscles [75], rigid skeletons [76], realistic materials [77], and accurate contact forces [78]. Our method also adopts this framework for faster and more stable time integration. In contrast to the aforementioned methods using Projective Dynamics, our algorithm is the first to enable joint actuation in articulated skeletons together with a generalized dry frictional contact for soft body dynamics.

1.2.3 Differentiable 3D Reconstruction

Differentiable volume rendering with implicit neural representation have recently been used to synthesize novel views, estimate geometry and appearance. Neural Radiance Field (NeRF) [79], and its follow-on works [80, 81, 82, 83] have achieved photorealistic rendering results even on complex scenes. The expressive coordinate-based multi-layer perceptron (MLP) representation equips the scene with high resolution [84] while remains regularized [85] and memory-efficient. Compared with differentiable surface rendering [86, 87], volume rendering could backpropagate gradients to the entire space so to better handle complex geometry optimization with topological changes and mitigate local minima. The original density field from NeRF cannot directly reconstruct high-quality surface mesh, so SDF-based neural fields [88, 89, 90] are proposed for multi-view geometry reconstruction. Compared to other discrete representation like mesh [91, 92, 93], point cloud [94, 95] and voxels [96, 97], SDF-field has higher resolution. And the the NeRF-based method can take raw RGB images without pixel-level segmentation [98, 99, 100]. However, current SDF-based neural fields cannot handle dynamic scenes. Recent work has extended NeRF to videos of dynamic scenes [101, 102, 103], but they mainly focus on the 2D image synthesis, and few of them can reconstruct the 3D geometry well. They either learn a canonical field with time-dependent motion [104, 105, 106, 107] or condition all the fields on the time axis [108, 109, 110]. We choose the former strategy because decoupling motion from the geometry makes the formulation more constrained and it also provides us with point-wise correspondence. All the mentioned dynamic extension of NeRF focus on the 2D video reconstruction and lack regularization on the 3D geometry. Our method can reconstruct the geometry much better than the existing works. Moreover, since single-video 3D reconstruction

is highly ill-posed, our geometry prior can also help improving the image reconstruction quality.

1.3 Outline of Dissertation

The subsequent chapters in this dissertation are organized as follows.

In Part II, we will demonstrate how to develop differentiable simulation for several dynamical systems.

Chapter 2 is dedicated to the two-way coupling of rigid body and cloth. We develop a scalable framework for differentiable physics that can support a large number of objects and their interactions. To accommodate objects with arbitrary geometry and topology, we adopt meshes as our representation and leverage the sparsity of contacts for scalable differentiable collision handling. Collisions are resolved in localized regions to minimize the number of optimization variables even when the number of simulated objects is high. We further accelerate implicit differentiation of optimization with nonlinear constraints. Experiments demonstrate that the presented framework requires up to two orders of magnitude less memory and computation in comparison to recent particle-based methods. We further validate the approach on inverse problems and control scenarios, where it outperforms derivative-free and model-free baselines by at least an order of magnitude.

Chapter 3 utilizes the adjoint method and checkpoint method for high-performance articulated body simulation. We present a method for efficient differentiable simulation of articulated bodies. This enables integration of articulated body dynamics into deep learning frameworks, and gradient-based optimization of neural networks that operate on articulated bodies. We derive the gradients of the forward dynamics using spatial algebra and the adjoint method. Our approach is an order

of magnitude faster than autodiff tools. By only saving the initial states throughout the simulation process, our method reduces memory requirements by two orders of magnitude. We demonstrate the utility of efficient differentiable dynamics for articulated bodies in a variety of applications. We show that reinforcement learning with articulated systems can be accelerated using gradients provided by our method. In applications to control and inverse problems, gradient-based optimization enabled by our work accelerates convergence by more than an order of magnitude.

Chapter 4 provides a comprehensive simulation platform for articulated soft bodies. We present a method for differentiable simulation of soft articulated bodies. Our work enables the integration of differentiable physical dynamics into gradient-based pipelines. We develop a top-down matrix assembly algorithm within Projective Dynamics and derive a generalized dry friction model for soft continuum using a new matrix splitting strategy. We derive a differentiable control framework for soft articulated bodies driven by muscles, joint torques, or pneumatic tubes. The experiments demonstrate that our designs make soft body simulation more stable and realistic compared to other frameworks. Our method accelerates the solution of system identification problems by more than an order of magnitude, and enables efficient gradient-based learning of motion control with soft robots.

Chapter 5 presents other differentiable systems that I have been involved with, including traffic simulation, fluid simulation, and quantum computing.

In Part III, we will go further and study how to fuse simulation and real-world observation.

Chapter 6 proposes to combine NeRF and differentiable simulation to reconstruct and simulate video data. We present a method for learning 3D geometry and physics parameters of a dynamic scene from only a monocular RGB video input. To decouple the learning of underlying scene geometry from dynamic motion, we represent the scene as a time-invariant

signed distance function (SDF) which serves as a reference frame, along with a time-conditioned deformation field. We further bridge this neural geometry representation with a differentiable physics simulator by designing a two-way conversion between the neural field and its corresponding hexahedral mesh, enabling us to estimate physics parameters from the source video by minimizing a cycle consistency loss. Our method also allows a user to interactively edit 3D objects from the source video by modifying the recovered hexahedral mesh, and propagating the operation back to the neural field representation. Experiments show that our method achieves superior mesh and video reconstruction of dynamic scenes compared to competing Neural Field approaches, and we provide extensive examples which demonstrate its ability to extract useful 3D representations from videos captured with consumer-grade cameras.

Chapter 7 designs a two-way coupling between mesh and NeRF during the rendering and simulation. We first review the light transport equations for both mesh and NeRF, then distill them into a straightforward algorithm for updating radiance and throughput along a cast ray with an arbitrary number of bounces. To address the discrepancy between the linear color space that the path tracer assumes, versus the sRGB color space that standard NeRF uses, we train NeRF with High Dynamic Range images to pull the radiance field back to the linear space. Finally, we consider how the hybrid surface-volumetric formulation can be efficiently integrated with a high-performance physics simulator that supports rigid and soft bodies, and cloth. The full rendering and simulation system can be run on a GPU at interactive rates. We show that a hybrid system approach outperforms alternatives in achieving visually realistic mesh insertion into a NeRF scene, because it allows realistic light transport from volumetric NeRF media onto surfaces, which affects the appearance of reflective/refractive surfaces and illumination of diffuse surfaces informed by the scene.

Chapter 8 introduces a method to utilize differentiable priors in hand-related tasks. Our approach uses rendering priors to align with input images and segmentation masks, while physics priors mitigate penetration and relative-sliding across frames. Furthermore, we present two alternatives for hand and object pose estimation. The optimization-based pose estimation achieves high accuracy, while the filtering-based tracking, which utilizes the differentiable priors as dynamics and observation models, runs faster. We show that our method achieves comparable or better results in the pose estimation task, and the differentiable physics module enables us to predict contact information for pose refinement. We also show that our approach generalizes to perception tasks including robotic hand manipulation and human-object pose estimation in the wild.

Chapter 9 includes two additional works on simulation from real-world observations.

Chapter 10 concludes by summarizing the key contributions and suggest possible future directions in extending my dissertation research.

Part II

Differentiable Simulation

Chapter 2: Scalable Differentiable Physics for Learning and Control

2.1 Introduction

Differentiable physics enables gradient-based learning systems to strictly adhere to physical dynamics. By making physics simulation differentiable, we can backpropagate through the physical consequences of actions. This enables agents to quickly learn to achieve desired effects in the physical world. It is also a principled and effective approach to inverse problems that involve physical systems [1, 2, 14, 39].

Recent efforts have significantly advanced the understanding of differentiable physics in machine learning and robotics. Automatic differentiation and analytical methods have been applied to derive a variety of differentiable simulation engines [3, 5, 19, 32]. Yet existing differentiable frameworks are still limited in their ability to simulate complex scenes composed of many detailed interacting objects. For example, modeling assumptions have limited some existing frameworks to restricted object classes, such as balls or two-dimensional polygons [1, 2]. Other approaches adopt expressive grid- or particle-based formulations, which are general in principle but have not scaled to detailed simulation of large scenes due to the cubic growth rates of volumetric grids and particles [6, 38, 111]. For example, a grid of size 1024^3 has on the order of a billion elements, and this resolution is still quite coarse in comparison to the scale and detail of physical environments we encounter in daily life, such as city streets. Particle-based

representations likewise suffer from explosive growth rates or limited resolution: Macklin et al. [112] suggest a maximum size ratio of 1:10 between the smallest and largest feature, or placing particles only around boundaries, which runs the risk of object tunneling.

In this chapter, we develop a differentiable physics framework that addresses these limitations. We adopt *meshes* as a general representation of objects. Meshes are the most widely used specification for object geometry in computer graphics and scientific computing. Meshes are inherently sparse, can model objects of any shape, and can compactly specify environments with both large spatial extent and highly detailed features [113].

The use of meshes brings up the challenge of collision handling, since collisions can occur anywhere on the surface of the mesh. Prior frameworks adopted global LCP-based formulations, which severely limited scalability [1, 2]. In contrast, we leverage the structure of contacts by grouping them into localized impact zones, which can be efficiently represented and processed [114, 115]. This substantially reduces the number of variables and constraints involved in specifying dynamical scenes and dramatically speeds up backpropagation.

Following prior work, we use implicit differentiation to compute gradients induced by an optimization problem embedded in the simulation [3, 116]. One of our contributions is an acceleration scheme that can handle the nonlinear constraints encountered in our system.

We evaluate the scalability and generality of the presented approach experimentally. Controlled experiments that vary the number of objects in the scene and their relative scales demonstrate that our approach is dramatically more efficient than state-of-the-art differentiable physics frameworks. Since meshes can represent both rigid and deformable objects, our work is also the first differentiable physics framework that can simulate two-way coupling of rigid bodies and cloth. We further demonstrate example applications to learning and control scenarios, where the presented framework

outperforms derivative-free and model-free baselines by at least an order of magnitude. Code is available on our project page: <https://gamma.umd.edu/researchdirections/mlphysics/diffsim>

2.2 Related Work

Backpropagating through physical dynamics is an enticing prospect for embedding physical reasoning into learning processes. Degraeve et al. [1] articulated the potential of this approach and applied automatic differentiation to rigid body systems where contacts are assumed to only happen between balls and planes. Although this assumption simplifies collision detection and response, it greatly limits the scope of application. de Avila Belbute-Peres et al. [2] introduced implicit differentiation for gradient computation, obviating the need to explicitly differentiate through all steps of the forward optimization solver. By designing specific rules for collision response, their framework supports simple two-dimensional shapes such as circles and polygons, but was not extended to general 3D objects. Toussaint et al. [14] utilized differentiable primitives to develop a robot reasoning system that can achieve user-defined tasks. It is designed for path planning with sphere-swept shapes and frictionless interactions, but not for general control tasks.

Building on the implicit differentiation methodology, Liang et al. [3] developed a differentiable simulator for cloth. Their formulation assumes that each node can move independently and is only tied to others by soft constraints. This is true for cloth but not for rigid bodies. Carpentier and Mansard [117] developed algorithms for efficient computation of analytical derivatives of rigid-body dynamics, but did not handle collision. Millard et al. [118] implemented an articulated body simulator in a reverse-mode automatic differentiation framework, but likewise did not handle

dynamic contact and collision response, which are at the heart of our work.

Hu et al. [6] developed a differentiable simulator for deformable objects based on the material point method (MPM). A follow-up work applies this framework to controller design for soft robots [9]. This approach is effective in handling soft bodies, but has limited ability to enforce rigid structure. Since the method is based on particles and grids, it has limited scalability and has not been applied to large scenes with interacting objects. In contrast, our method uses meshes to represent objects, naturally enforces precise detailed geometry, and scales to scenes with both large spatial extent and fine local features.

Hu et al. [19] introduce a domain-specific language for building differentiable physical simulators. This work develops a programming language and compiler that can be broadly useful throughout the differentiable physics community, including to projects such as ours. However, at present this framework does not deal with general mesh-based rigid and deformable objects. Their rigid-body scenario uses rectangles constrained by springs, with contact responses only possible between rectangles and the ground plane. Their three-dimensional scenario uses soft bodies modeled by particles and grids and inherits the aforementioned limitations of MPM-based modeling.

A line of work developed particle-based frameworks for differentiable simulation of fluids [39, 44] and unified treatment of fluids and solids [38, 111]. These frameworks resort to approximations in handling rigid bodies and detailed geometry, and suffer from high growth rates in the scale of objects and scenes. The generality of particle-based approaches is appealing, but their scalability limitations are well-known and are explicitly discussed in the literature [112]. We take a different tack and model objects as meshes, thus harnessing the scalability and generality of this representation.

Some of the aforementioned works, as well as others [36, 37, 40, 42], fit function approximators

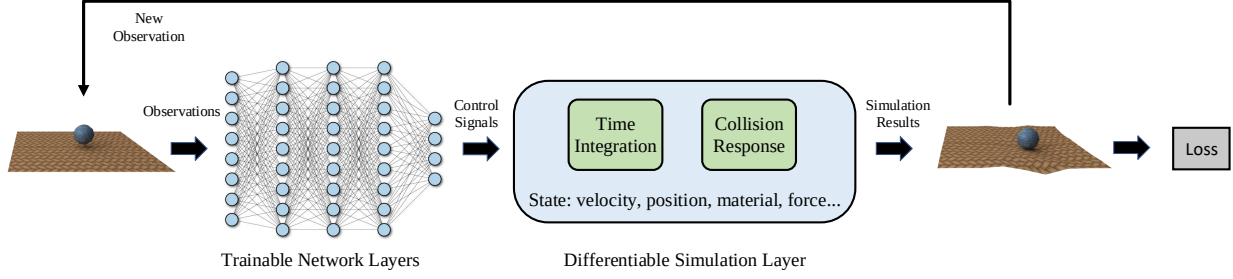


Figure 2.1: Differentiable simulation embedded in a neural network. The loss can be backpropagated through the physics simulator to the control signal, thus enabling end-to-end gradient-based training of the controller network. This approach enables faster convergence with higher accuracy in learning and control of physical systems.

to demonstrations of physical dynamics. Such approximators are valuable, but may not fully capture the true underlying physics and may degenerate outside the training distribution. Our work follows the program exemplified by Degraeve et al. [1], de Avila Belbute-Peres et al. [2], and others in that we aim to backpropagate through the true physical dynamics. In this approach, physical correctness is enforced by construction.

2.3 Overview

Our differentiable physics engine can function as a layer embedded within a broader differentiable pipeline, with other neural network layers in the loop. This scenario is illustrated in Figure 2.1. Given observations from simulation, the network can compute control signals and send them to the simulator. The physics engine performs time integration, evaluates collision response, and updates the mesh states. The simulation results can be used to evaluate the loss and provide new observations to the network, thus closing the loop. All operations in our physics engine are differentiable, supporting end-to-end gradient-based optimization of the complete pipeline. As mentioned earlier, state-of-the-art differentiable simulation algorithms often suffer from scalability issues. They either use particle representations, which have cubic growth rates with respect to

object sizes, or compute custom dynamic responses for a limited number of object primitives, which have limited ability to represent complex shapes. Our method resolves these issues to advance scalability and generality.

Section 2.4 introduces our notation and the basic forward integration scheme. To facilitate scalability and generality, we adopt meshes as our basic object representation. Simulation of complex scenes brings up the challenge of collision detection and response. We describe an efficient collision handling scheme in Section 2.5. Observing that collisions are often sparse, we utilize the impact zone method and handle collisions and contact forces locally in independent collision areas. This collision handling scheme has linear computational complexity with respect to the number of constraints rather than the total number of nodes. Our unified collision handling formulation also enables differentiable two-way coupling of rigid bodies and cloth. In Section 2.6, we describe the challenges involved in differentiation and introduce an acceleration scheme for implicit differentiation with nonlinear constraints.

2.4 Preliminaries

After construction of the system, the dynamics can be written as

$$\frac{d}{dt} \begin{pmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{pmatrix} = \begin{pmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{pmatrix} = \begin{pmatrix} \dot{\mathbf{q}} \\ \mathbf{M}^{-1} \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) \end{pmatrix}, \quad (2.4.0.1)$$

where $\mathbf{M}$ is the mass matrix and $\mathbf{f}$ is the generalized force vector. We incorporate contact forces [115], internal forces including bending and stretching [119], and external forces including gravity and control input. For numerical simulation, the dynamical system can be discretized with

time step h . At time t_0 , with $\mathbf{q}_0 = \mathbf{q}(t_0)$ and $\dot{\mathbf{q}}_0 = \dot{\mathbf{q}}(t_0)$, we can compute the increment of the coordinates $\Delta\mathbf{q} = \mathbf{q}(t_0 + h) - \mathbf{q}(t_0)$ and velocities $\Delta\dot{\mathbf{q}} = \dot{\mathbf{q}}(t_0 + h) - \dot{\mathbf{q}}(t_0)$ using the implicit Euler method [120]:

$$\begin{pmatrix} \Delta\mathbf{q} \\ \Delta\dot{\mathbf{q}} \end{pmatrix} = h \begin{pmatrix} \dot{\mathbf{q}}_0 + \Delta\dot{\mathbf{q}} \\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{q} + \Delta\mathbf{q}, \dot{\mathbf{q}} + \Delta\dot{\mathbf{q}}) \end{pmatrix}. \quad (2.4.0.2)$$

With a linear approximation, we can solve for $\Delta\dot{\mathbf{q}}$,

$$\left(h^{-1}\mathbf{M} - \frac{\partial\mathbf{f}}{\partial\dot{\mathbf{q}}} - h\frac{\partial\mathbf{f}}{\partial\mathbf{q}} \right) \Delta\dot{\mathbf{q}} = \mathbf{f}_0 + h\frac{\partial\mathbf{f}}{\partial\mathbf{q}}\dot{\mathbf{q}}_0, \quad (2.4.0.3)$$

which can then be used to compute $\mathbf{q}(t_0 + h)$ and $\dot{\mathbf{q}}(t_0 + h)$ for the next time step.

2.5 Scalable Collision Handling

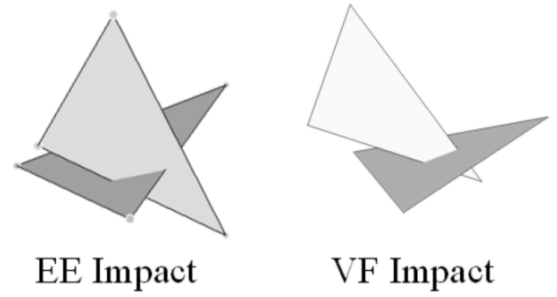
One of the most time-consuming and complex parts of physics simulation is collision handling. The difficulty, especially for differentiable physics, lies in how to compute the correct derivatives efficiently even in a large scene with many interacting objects. As observed by Hu et al. [19], naive discrete-time impulse-based collision response can lead to completely incorrect gradients. We apply continuous collision detection to circumvent this problem. We also employ a bounding volume hierarchy to localize and accelerate dynamic collision detection.

Traditional collision response algorithms for rigid bodies commonly employ global LCP-based solvers to resolve collisions [2]. This works well for small-to-medium-size systems, but does not scale well when the number of objects increases. The problem is exacerbated when the derivatives of all DOFs in this large system need to be computed simultaneously, which is exactly

what backpropagation does.

Our key observation is that collisions are sparse in the majority of simulation steps, which means that the number of collisions is much smaller than the total DOF. As shown in Figure 2.2, collisions are commonly localized around the scene. With this observation, we resolve collision locally among the objects involved, rather than setting up a global optimization, updating coordinates, and computing the gradients for all objects in the environment.

One way to represent local collisions is to use impact zones [114, 115]. An impact is a pair of primitives colliding with each other. It can be an edge-edge (EE) or a vertex-face (VF) pair. For these two cases, the non-penetration constraints can



be written as

$$\begin{aligned}
0 \leq C_{ee} &= \mathbf{n} \cdot [(\alpha_3 \mathbf{x}_3 + \alpha_4 \mathbf{x}_4) - (\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2)] \\
0 \leq C_{vf} &= \mathbf{n} \cdot [\mathbf{x}_4 - (\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \alpha_3 \mathbf{x}_3)],
\end{aligned} \tag{2.5.0.1}$$

respectively, where $\mathbf{x}_i$ are the positions of the mesh vertices, α_i are the barycentric coordinates, and $\mathbf{n}$ is the normal of the colliding plane.

Impacts may share vertices. All the impacts in one connected component are said to form an impact zone. Each impact zone is a local area that can be treated independently. The red areas in Figure 2.2 are impact zones. Collision resolution in one impact zone can be written as

an optimization problem:

$$\begin{aligned} & \underset{\mathbf{x}'}{\text{minimize}} && \frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \mathbf{M}(\mathbf{x} - \mathbf{x}') \\ & \text{subject to} && \mathbf{G}\mathbf{x}' + \mathbf{h} \leq \mathbf{0}. \end{aligned} \tag{2.5.0.2}$$

Here $\mathbf{x}$ is the concatenation of vertex positions in the impact zone before collision handling; $\mathbf{x}'$ are the resolved positions that satisfy constraints; $\mathbf{M}$ is the mass matrix; $\mathbf{G}$ and $\mathbf{h}$ are a matrix and a vector derived from the constraints in Equation 2.5.0.1.

The impact zone method is fail-safe [114] and is commonly used in cloth simulation, where each vertex in the cloth is free and can be explicitly optimized when solving Equation 2.5.0.2. However, we cannot directly use this formulation for rigid bodies, because all vertices in a rigid body are tied and cannot be optimized separately. A possible treatment is to use additional constraints on the relative positions of vertices to enforce rigidity. However, the number of vertices in a rigid body is usually far larger than its DOF. We therefore take a different approach that minimizes the number of optimization variables in the system. This is critical for scalable backpropagation in such systems.

We make a distinction between contact vertices $\mathbf{x}$ and the objects' generalized coordinates $\mathbf{q}$. The actual optimization variables are replaced by $\mathbf{q}$. For a rigid body i with rotation $\mathbf{r} = (\phi, \theta, \psi)^\top$ and translation $\mathbf{t} = (t_x, t_y, t_z)^\top$, the generalized coordinates are $\mathbf{q} = [\mathbf{r}^\top, \mathbf{t}^\top]^\top \in \mathbb{R}^6$. The block-diagonal mass matrix $\hat{\mathbf{M}} \in \mathbb{R}^{6 \times 6}$ is accordingly changed into angular and linear inertia. The constraints are rewritten using the new variables. A function $\mathbf{f}(\cdot)$ maps generalized coordinates $\mathbf{q}$ to a contact point $\mathbf{x}$, including both rotation and translation. Detailed formulations of $\hat{\mathbf{M}}$ and $\mathbf{f}(\cdot)$ are given in Appendices A and A. The optimization problem for collision resolution

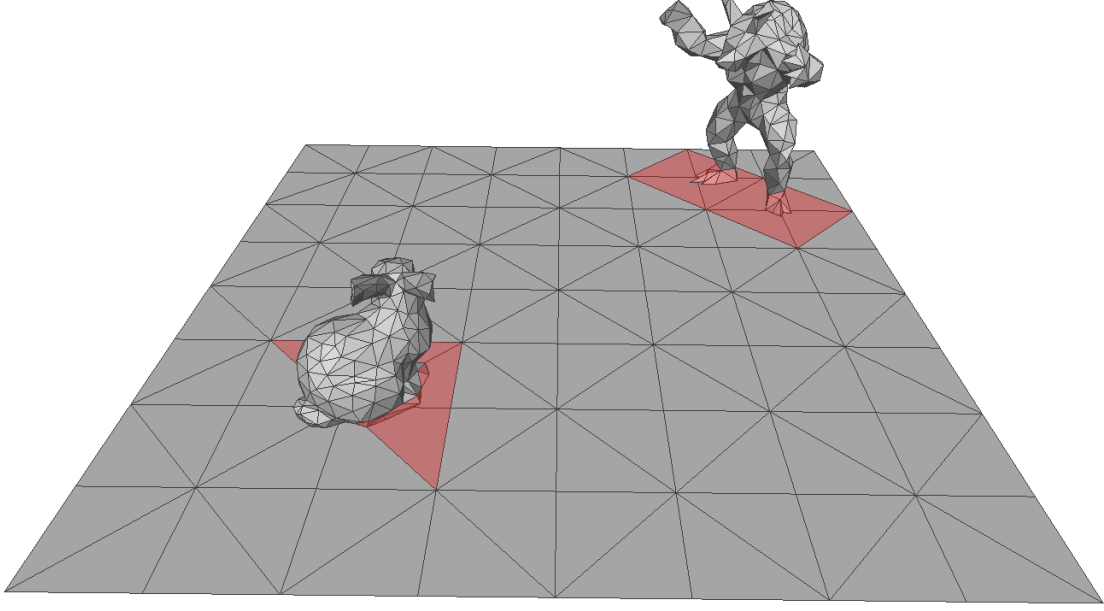


Figure 2.2: Collision visualization. Regions in collision are shown in red. In most cases, the number of collisions is sparse compared to the number of degrees of freedom in the scene.

becomes

$$\begin{aligned}
 & \underset{\mathbf{q}'}{\text{minimize}} && \frac{1}{2}(\mathbf{q} - \mathbf{q}')^\top \hat{\mathbf{M}}(\mathbf{q} - \mathbf{q}') \\
 & \text{subject to} && \mathbf{G}\mathbf{f}(\mathbf{q}') + \mathbf{h} \leq \mathbf{0}.
 \end{aligned} \tag{2.5.0.3}$$

2.6 Fast Differentiation

Gradients for the sparse linear system in Equation 2.4.0.3 can be computed via implicit differentiation [3, 116]. Other basic operations can be handled by automatic differentiation [121]. However, the heaviest computation is induced by collision handling, which can take up to 90% of the runtime for a single simulation step. Collision resolution can involve multiple iterations of the optimization specified in Equation 2.5.0.3 for each impact zone. Liang et al. [3] proposed

a method that reduces computation, but it only deals with linear constraints and does not apply in our case due to the nonlinearity introduced by $\mathbf{f}(\cdot)$. We now develop an appropriate acceleration scheme that works in our setting.

Consider the differentiation in our case. At a local minimum $\mathbf{z}^*$ with Lagrange multiplier λ^* , the following KKT conditions hold:

$$\begin{aligned}\hat{\mathbf{M}}\mathbf{z}^* - \hat{\mathbf{M}}\mathbf{q} + \nabla \mathbf{f}^\top \mathbf{G}^\top \lambda^* &= 0 \\ D(\lambda^*)(\mathbf{G}\mathbf{f}(\mathbf{z}^*) + \mathbf{h}) &= 0,\end{aligned}\tag{2.6.0.1}$$

where $D(\cdot)$ packages a vector into a diagonal matrix. The key idea here is to linearize $\mathbf{f}(\cdot)$ around a small neighborhood of $\mathbf{z}^*$. The implicit differentiation is formulated as

$$\begin{aligned}\begin{bmatrix} \hat{\mathbf{M}} & \nabla \mathbf{f}^\top \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G}\nabla \mathbf{f} & D(\mathbf{G}\mathbf{f}(\mathbf{z}^*) + \mathbf{h}) \end{bmatrix} \begin{bmatrix} \partial \mathbf{z} \\ \partial \lambda \end{bmatrix} &= \\ \begin{bmatrix} \hat{\mathbf{M}}\partial \mathbf{q} - \nabla \mathbf{f}^\top \partial \mathbf{G}^\top \lambda^* \\ -D(\lambda^*)(\partial \mathbf{G} \cdot \mathbf{f}(\mathbf{z}^*) + \partial \mathbf{h}) \end{bmatrix} &.\end{aligned}\tag{2.6.0.2}$$

As derived by Amos and Kolter [116], by solving for $\mathbf{d}_z, \mathbf{d}_\lambda$ in the following equation

$$\begin{bmatrix} \hat{\mathbf{M}} & \nabla \mathbf{f}^\top \mathbf{G}^\top D(\lambda^*) \\ \mathbf{G}\nabla \mathbf{f} & D(\mathbf{G}\mathbf{f}(\mathbf{z}^*) + \mathbf{h}) \end{bmatrix} \begin{bmatrix} \mathbf{d}_z \\ \mathbf{d}_\lambda \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{z}}^\top \\ \mathbf{0} \end{bmatrix}\tag{2.6.0.3}$$

we can get the backward gradients:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}} = \mathbf{d}_z^\top \hat{\mathbf{M}} \quad (2.6.0.4)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{G}} = -D(\lambda^*) \mathbf{d}_\lambda \mathbf{f}(\mathbf{z}^*)^\top - \lambda^* \mathbf{d}_z^\top \nabla \mathbf{f}^\top \quad (2.6.0.5)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}} = -\mathbf{d}_\lambda^\top D(\lambda^*). \quad (2.6.0.6)$$

Assume that in one impact zone, there are n DOFs and m constraints. In Equation 2.6.0.3, the size of the system is $n + m$. The solution of this system would typically take $O((n + m)^3)$: prohibitively expensive in large scenes. We therefore need to accelerate the computation of $\mathbf{d}_z$ and $\mathbf{d}_\lambda$. Liang et al. [3] proposed to use a QR decomposition. However, it only copes with quadratic programming with linear constraints. We can extend this method to nonlinear constraints by incorporating the Jacobian $\nabla \mathbf{f}$.

Recall that a rigid body has coordinates $\mathbf{q} = [\mathbf{r}, \mathbf{t}] \in \mathbb{R}^6$. The rotation is represented by Euler angles $\mathbf{r} = (\phi, \theta, \psi)$. The Jacobian at vertex $\mathbf{x} = (x, y, z)^\top$ of the rigid body is

$$\nabla \mathbf{f}_k = \begin{bmatrix} \frac{\partial x}{\partial \phi} & \frac{\partial x}{\partial \theta} & \frac{\partial x}{\partial \psi} & 1 & 0 & 0 \\ \frac{\partial y}{\partial \phi} & \frac{\partial y}{\partial \theta} & \frac{\partial y}{\partial \psi} & 0 & 1 & 0 \\ \frac{\partial z}{\partial \phi} & \frac{\partial z}{\partial \theta} & \frac{\partial z}{\partial \psi} & 0 & 0 & 1 \end{bmatrix}. \quad (2.6.0.7)$$

The exact formulation of $\nabla \mathbf{f}_k$ can be found in Appendix A.

After computing $\nabla \mathbf{f}$, we apply a QR decomposition $\sqrt{\hat{\mathbf{M}}}^{-1} \nabla \mathbf{f}^\top \mathbf{G}^\top = \mathbf{Q}\mathbf{R}$ and derive

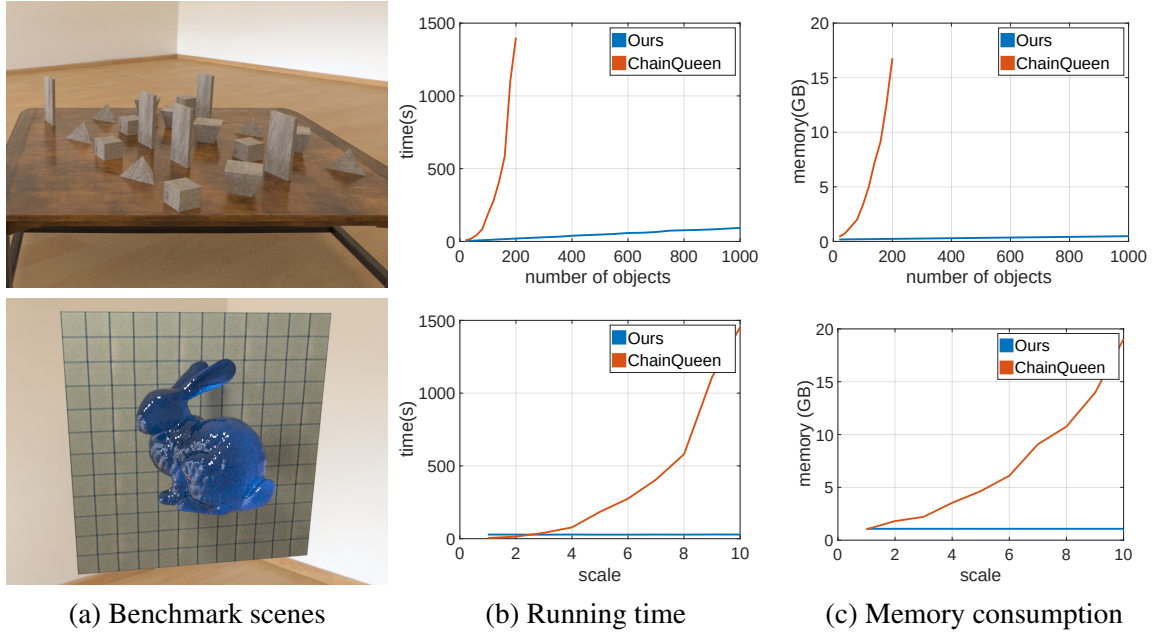


Figure 2.3: Scalability. (a) Benchmark scenes. Few differentiable simulation frameworks are capable of modeling these scenes. We compare to the expressive MPM-based framework ChainQueen [6], implemented in the high-performance DiffTaichi library [19]. (b,c) Runtime and memory consumption as the scenes are varied in controlled fashion. Memory consumption is peak memory usage. Time is the running time for simulating 2 seconds of dynamics. Top: the number of objects in the scene increases from 20 to 1000. Bottom: the relative scale of the cloth and the bunny increases from 1:1 to 10:1. The MPM-based framework consumes up to two orders of magnitude more memory and computation before it runs out of memory.

$\mathbf{d}_z, \mathbf{d}_\lambda$ from Equation 2.6.0.3 as

$$\mathbf{d}_z = \sqrt{\hat{\mathbf{M}}}^{-1} (\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top) \sqrt{\hat{\mathbf{M}}}^{-1} \frac{\partial \mathbf{L}}{\partial \mathbf{z}}^\top \quad (2.6.0.8)$$

$$\mathbf{d}_\lambda = D(\lambda^*)^{-1} \mathbf{R}^{-1} \mathbf{Q}^\top \sqrt{\hat{\mathbf{M}}}^{-1} \frac{\partial \mathbf{L}}{\partial \mathbf{z}}^\top. \quad (2.6.0.9)$$

The cost of the QR decomposition is $O(nm^2)$, reducing the computation from $O((n+m)^3)$.

2.7 Experiments

We conduct a variety of experiments to evaluate the presented approach. We begin by comparing to other differentiable physics frameworks, with particular attention to scalability and

generality. We then conduct ablation studies to evaluate the impact of the techniques presented in Sections 2.5 and 2.6. Lastly, we provide case studies that illustrate the application of the presented approach to inverse problems and learning control.

The automatic differentiation is implemented in PyTorch 1.3 [121]. All experiments are run on an Intel(R) Xeon(R) W-2123 CPU @ 3.60GHz.

2.7.1 Scalability

We construct two benchmark scenes and vary them in controlled fashion to evaluate scalability. The scenes are illustrated in Figure 2.3.

In the first, objects fall from the air, hit the ground, and finally settle. To test scalability, we increase the number of objects while maintaining the stride between objects. As the number of objects increases, the spatial extent of the scene expands accordingly.

In the second scene, a rigid bunny strikes a deformable cloth. We vary the relative scale of the bunny and the cloth to test the ability of simulation frameworks to handle objects with geometric features at different resolutions.

Few differentiable simulation frameworks are capable of modeling these scenes. de Avila Belbute-Peres et al. [2] only simulate 2D scenes with a restricted repertoire of primitives. Liang et al. [3] simulate cloth but cannot handle rigid-body dynamics. We therefore choose the simulation framework of Hu et al. [6] as our main baseline. This framework uses the material point method (MPM), which leverages particles and grids to model solids. We use the state-of-the-art implementation from the high-performance DiffTaichi library [19].

The results are shown in Figure 2.3. The first row reports the runtime and memory consumption

of the two frameworks as the number of objects in the scene increases from 20 to 1000. Memory consumption is peak memory usage. Time is the running time for simulating 2 seconds of dynamics. Our runtime and memory consumption increase linearly in the complexity of the scene, while the MPM-based method scales cubically until it runs out of memory at 200 objects and a 640^3 grid.

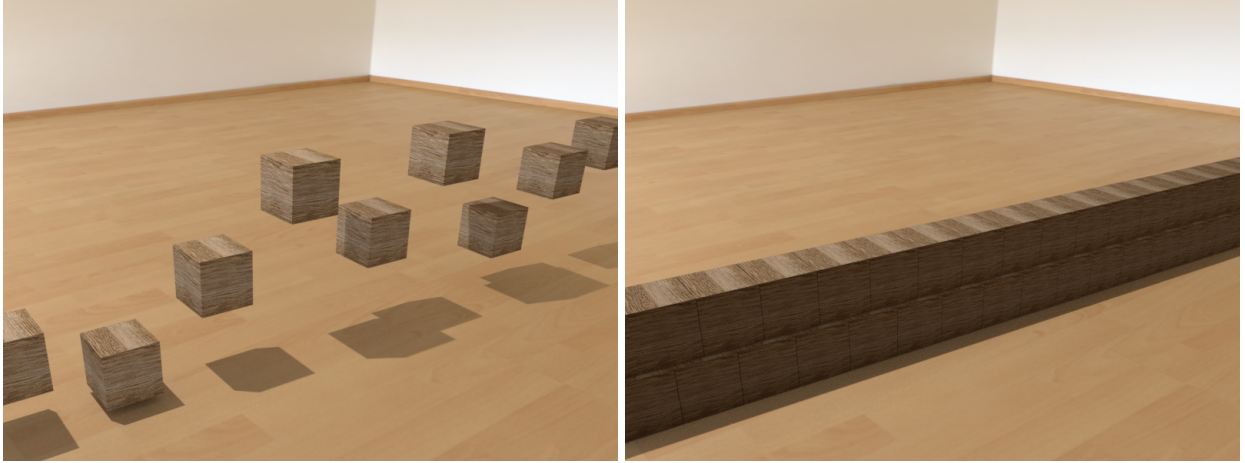
The second row reports runtime and memory consumption as the relative sizes of the cloth and the bunny are varied from 1:1 to 10:1. The runtime and memory consumption of our method stay constant. In contrast, as the size of the cloth grows, the MPM-based framework is forced to allocate more and more memory and expend greater and greater computation.

These experiments indicate that scalability is a significant advantage of our method. Since we do not need to quantize space, the extent of the scene or the relative sizes of objects do not dominate our runtime and memory consumption. Since our method dynamically detects and handles collisions locally, as needed, rather than setting up a global optimization problem, the runtime scales linearly (rather than quadratically) with scene complexity.

2.7.2 Ablation Studies

We proposed localized collision handling in Section 2.5 and fast differentiation in Section 2.6 to make our method scalable to large scenes with many degrees of freedom. We conduct dedicated ablation studies to assess the contribution of these techniques.

Localized collision handling. We compare with LCP-based rigid-body differentiable simulation developed by de Avila Belbute-Peres et al. [2]. This baseline also uses implicit differentiation to compute derivatives of optimization solvers [116]. The environment is shown in Figure 2.4(a).



(a) Localized collisions

(b) Correlated collisions

Figure 2.4: Ablation studies. (a) Comparing global and local collision handling. Cubes are released from the air and fall to the ground. (b) Evaluating the contribution of the fast differentiation scheme. Cubes are densely stacked, forming a big connected component. All collisions need to be solved simultaneously because motion of one cube can affect all others.

N ($= 100, 200, 300$) cubes are released above the ground plane. They fall down and hit the ground. We use the same environment for both methods, although the LCP-based framework only simulates in 2D, thus having only half the degrees of freedom compared to our 3D simulation (3 versus 6 per object, accounting for translation and rotation). We disabled our fast differentiation method in this experiment in order to neutralize its effect and conduct a controlled comparison between global and local collision handling.

The implementation of de Avila Belbute-Peres et al. [2] uses four threads while our method only uses one. Results are reported in Table 2.1. Our sparse collision handling method runs up to 5 times faster than the LCP-based approach, and the performance gap widens as the complexity of the scene increases.

Fast differentiation. We now evaluate the contribution of the acceleration scheme described in Section 2.6. The environment is shown in Figure 2.4(b). N ($= 100, 200, 300$) cubes are stacked in two layers. During collision handling, all contacts between cubes form one connected

# of cubes	100	200	300
LCP	0.73s $\pm$ 0.017s	2.87s $\pm$ 0.103s	8.42s $\pm$ 0.190s
Ours	0.56s $\pm$ 0.009s	1.11s $\pm$ 0.012s	1.65s $\pm$ 0.025s

Table 2.1: Runtime of backpropagation (in seconds per simulation step) with LCP-based collision handling [2] and our approach. Simulation of the scene shown in Figure 2.4(a). The LCP-based framework simulates in 2D and uses 4 threads, while our implementation simulates in 3D and uses 1 thread. Our approach is faster, and the performance gap widens with the complexity of the scene.

component. All constraints need to be solved in one big optimization problem. Thus the linear system in Equation 2.6.0.3 is large.

# of cubes	100	200	300
W/o FD	1.43s $\pm$ 0.015s	7.76s $\pm$ 0.302s	21.88s $\pm$ 0.125s
Ours	0.41s $\pm$ 0.004s	0.86s $\pm$ 0.008s	1.30s $\pm$ 0.008s
Speedup	3.49x	9.02x	16.83x

Table 2.2: Runtime of backpropagation (in seconds per simulation step) with and without our fast differentiation scheme. Simulation of the scene shown in Figure 2.4(b). N (= 100, 200, 300) cubes are stacked in two layers. The impact of the acceleration scheme increases with the complexity of the scene.

We evaluate the runtime of backpropagation in this scene with and without the presented acceleration scheme. The ablation condition is referred to as ‘W/o FD’, where FD refers to fast differentiation. In this condition, the derivative of Equation 2.5.0.3 is calculated by solving Equation 2.6.0.3 directly. The results are reported in Table 2.2. The fast differentiation accelerates backpropagation by up to 16x in this scene, and the impact of the technique increases with the complexity of the scene.

2.7.3 Two-way coupling

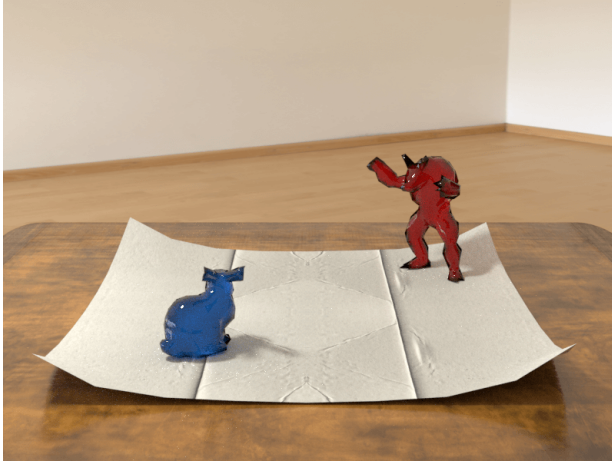
One of the advantages of this presented framework is that it can handle both rigid bodies and deformable objects such as cloth. This is illustrated in the two scenes shown in Figure 2.5. In Figure 2.5(a), a Stanford bunny and an armadillo stand on a piece of cloth. The cloth is lifted by its corners, envelopes the figurines, and lifts them up. In Figure 2.5(b), a piece of cloth strikes a domino, which begins a chain reaction that propagates until the last domino strikes the cloth from behind. The scenes are also shown in the supplementary video.

No prior differentiable simulation framework is capable of modeling these scenes properly. Liang et al. [3] can simulate the cloth, but its motion does not affect the figurines. The framework of Hu et al. [6, 19] does not enforce rigidity and suffers from interpenetration. In contrast, our approach simulates the correct dynamics and the two-way coupling between them.

We further compare with MuJoCo, a popular physics engine [122]. MuJoCo models cloth as a 2D grid of capsule and ellipsoid geoms in addition to spheres. This representation fails to correctly handle collisions near the holes in a grid. In Figure 2.6, we simulate a rigid body falling onto a trampoline and bouncing back. When simulated in MuJoCo, the ball penetrates the trampoline. Our method simulates the interaction correctly.

2.7.4 Applications

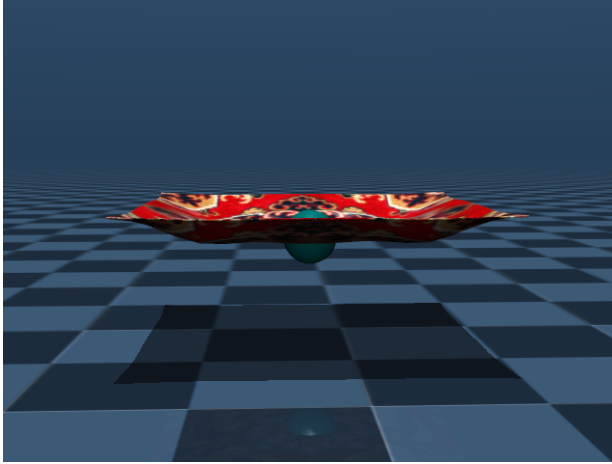
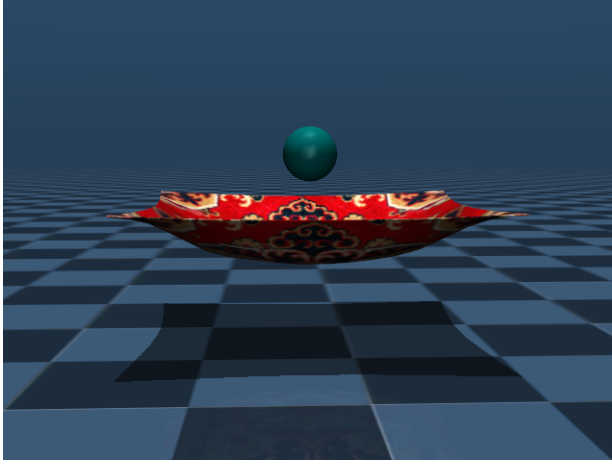
Inverse problem. Differentiable simulation naturally lends itself to gradient-based optimization for inverse problems. A case study is shown in Figure 2.7, in which a marble is supported by a soft sheet. The sheet is fixed at the four corners. In each time step, an external force is applied to the marble. The goal is to find a sequence of forces that drives the marble to a



(a) Bunny and armadillo

(b) Dominoes

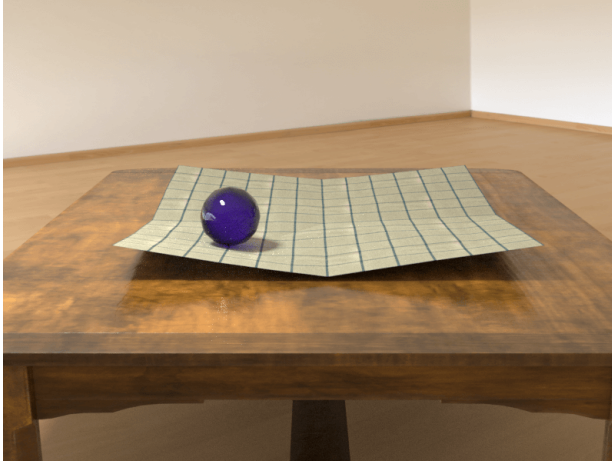
Figure 2.5: Two-way coupling of rigid bodies and cloth. Top: initial state. Bottom: towards the end of the simulation. (a) Two figurines are lifted by a piece of cloth. (b) Cloth strikes domino and is later struck back. No prior differentiable simulation framework is capable of modeling these scenes properly.



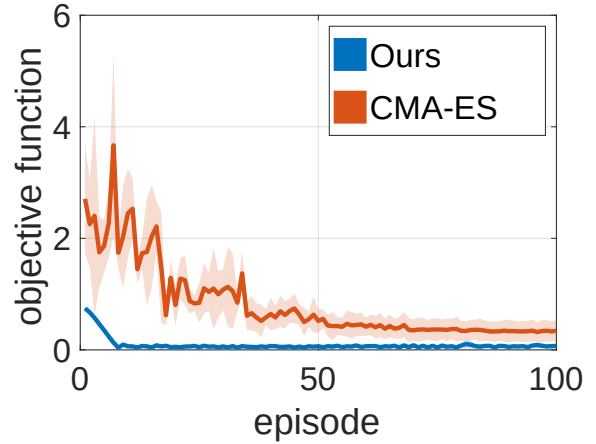
(a) MuJoCo

(b) Ours

Figure 2.6: Comparison with MuJoCo. Simulation of a ball interacting with a trampoline. Top: initial state. Bottom: later in the simulation. (a) Simulation in MuJoCo. (b) Simulation by our method. MuJoCo models cloth as a 2D grid of capsules and ellipsoid geoms in addition to spheres. The ball penetrates the trampoline when the grid is sparse.



(a) Marble on a sheet



(b) Objective function

Figure 2.7: Inverse problem. (a) The goal is to bring the marble to the desired position with minimal force. (b) Our framework enables gradient-based optimization for this problem and quickly converges to a lower objective value than derivative-free optimization (CMA-ES). We perform 5 runs with different random seeds; shaded areas represent one standard deviation.

target position in 2 seconds, while minimizing the total amount of applied force. The vertical component of the external force is set to 0 so that the marble has to interact with the cloth before reaching the target. We compare with a derivative-free optimization algorithm, CMA-ES [123]. Our framework enables gradient-based optimization in this setting and converges in 4 iterations, reaching a lower objective value than what CMA-ES achieves after two orders of magnitude more iterations.

Learning control. Our second set of case studies uses the differentiable simulator to backpropagate gradients in a neural network that must learn to manipulate objects to achieve desired outcomes. The scenarios are shown in Figure 2.8. The neural network controls a pair of sticks and a piece of cloth, respectively, which must be used to manipulate an object. The goal is to bring the object to the desired position within 1 second. The optimization objective is the L_2 distance to the target. The neural network is an MLP with 50 nodes in the first layer and 200 nodes in the second, with ReLU activations.

For reference, we report the performance of DDPG, a model-free reinforcement learning

algorithm [124]. In each episode, we fix the initial position of the manipulator and the object while the target position is randomized. For both methods, the input to the network is a concatenated vector of relative distance to the target, speed, and remaining time. The DDPG reward is the negative L_2 distance to the target.

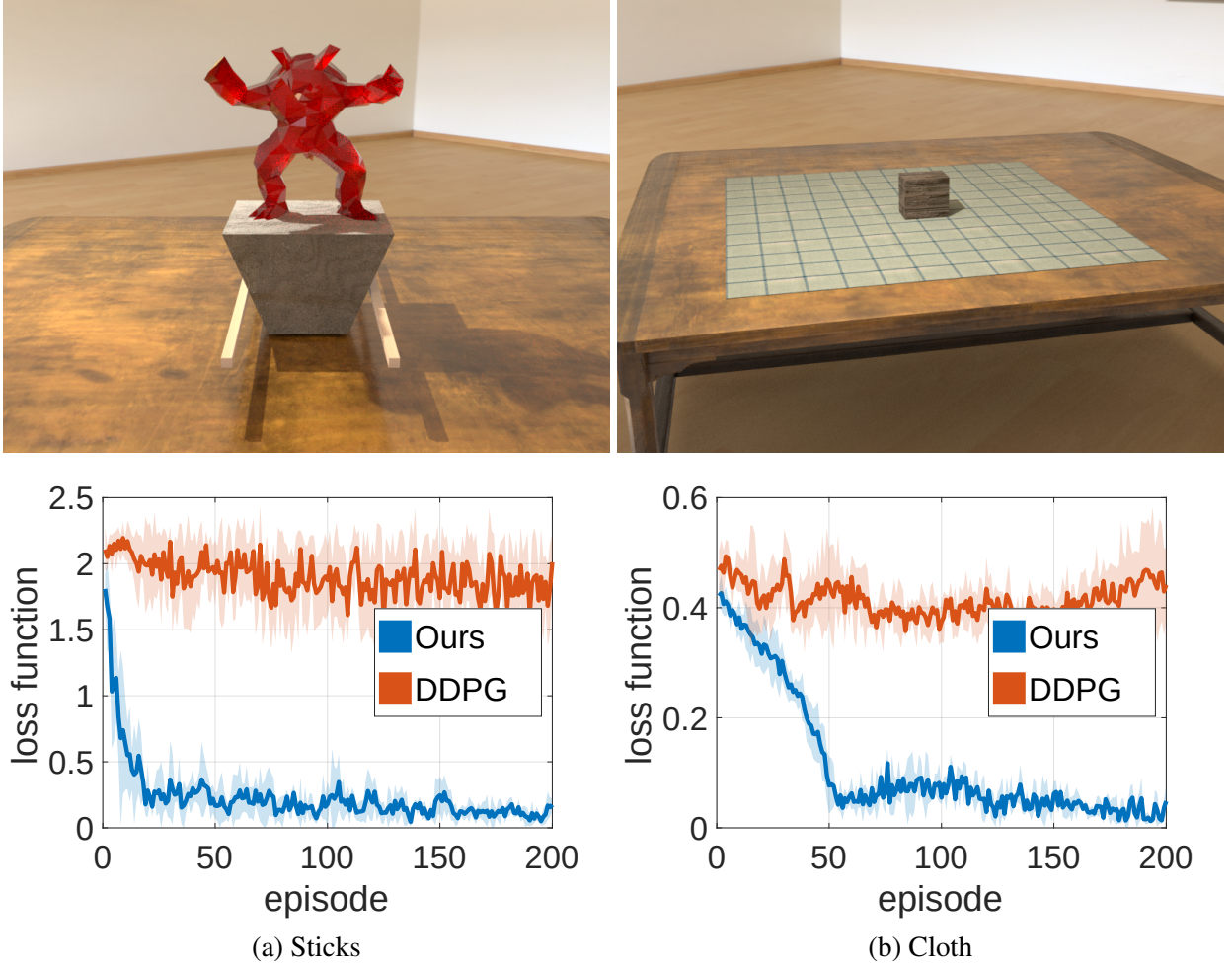


Figure 2.8: Learning control. A neural network must learn to control (a) a pair of sticks or (b) a piece of cloth. The trained controller must be able to bring the manipulated object to any desired location specified at test time. Our method enables gradient-based training of the controller. A model-free reinforcement learning baseline (DDPG) fails to learn the task on a comparable time scale. We perform 5 runs with different random seeds; shaded areas represent one standard deviation.

Our method updates the network once at the end of each episode, while DDPG receives a reward signal and updates the network weights in each time step. As shown in the loss curves

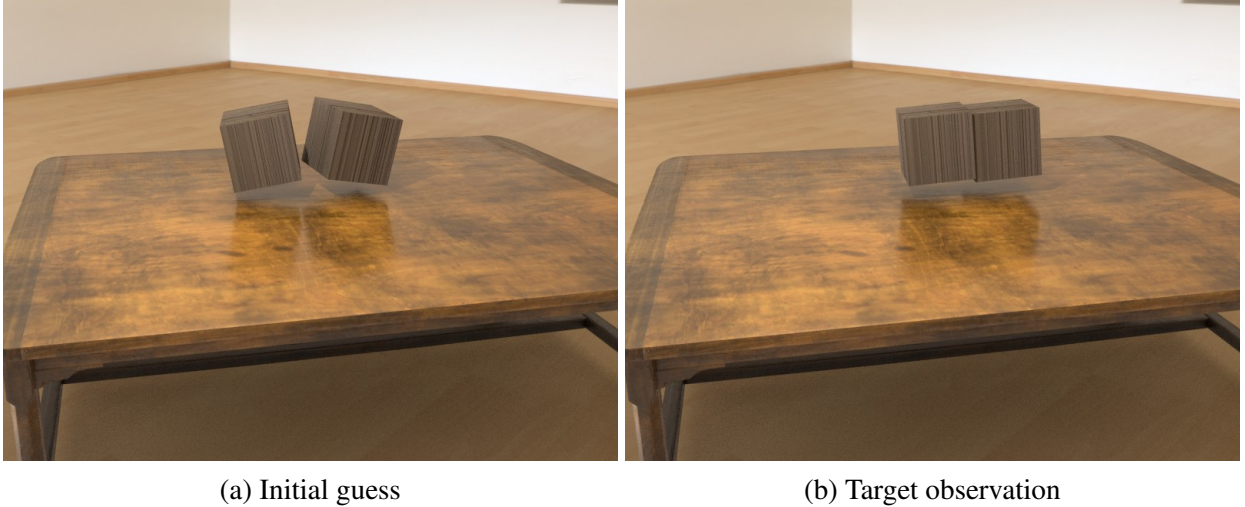


Figure 2.9: Parameter estimation. Cubes start with opposite velocities and collide with each other. Our goal is to estimate the mass of the left cube such that the total momentum after the collision has the desired direction and magnitude. (a) is the initial estimate: both cubes have the same mass and the total momentum after collision is 0. (b) is the simulation with the mass produced by our estimation procedure: the momentum points to the right and has a magnitude of 3, as specified in the target observation.

in Figure 2.8, our method quickly converges to a good control policy, while DDPG fails to learn the task on a comparable time scale. This example illustrates the power of gradient-based optimization with our differentiable physics framework.

Parameter estimation. Given an observation of motion, our method can estimate unknown physical parameters for objects in the scene.

In Figure 2.9, two cubes start with opposite velocities $\pm \mathbf{v}$ and collide with each other. Given a target momentum observation $\mathbf{p} = m_1 \mathbf{v}'_1 + m_2 \mathbf{v}'_2$, where $\mathbf{v}'_{1,2}$ are velocities after collision and $m_{1,2}$ are masses, we aim to estimate the mass m_1 . The target momentum is set to $\mathbf{p} = (3, 0, 0)$. We initialize with $m_1 = m_2 = 1$ and $\mathbf{p} = (0, 0, 0)$ (Figure 2.9(a)). After 90 gradient steps, our method arrives at the estimate $m_1 = 5.4$ and achieves the desired momentum (Figure 2.9(b)).

Interoperability across simulators. Our differentiable simulation framework is interoperable with other physics simulators. We have evaluated this interoperability with MuJoCo, a popular

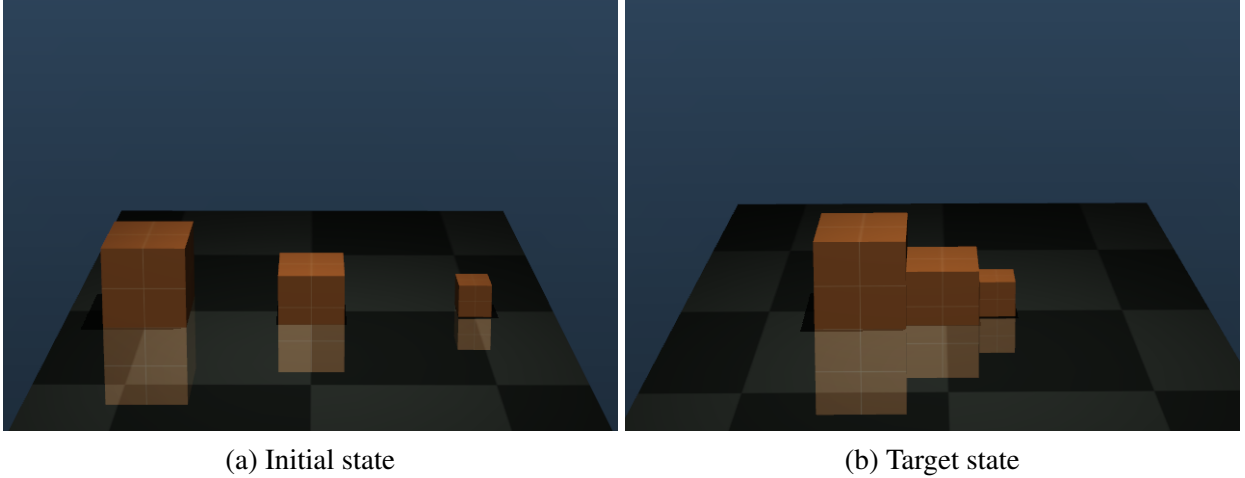


Figure 2.10: Interoperability across simulators. Three cubes are placed on smooth ground. The goal is to make the cubes stick together while minimizing the applied forces. The loss is computed in MuJoCo but the gradient is evaluated in DiffSim. (a) is the initial state and (b) is the successful simulation result after the optimization.

non-differentiable simulator [122]. The experiment is illustrated in Figure 2.10. We place three cubes on smooth ground. The goal is to apply forces to the cubes in order to make them stick together. The objective function is the distance of each cube to its target positions plus the L_2 norm of the force. We compute the loss in MuJoCo but evaluate the gradient in DiffSim, our differentiable simulator. Figure 2.10(a) shows the initial state. Figure 2.10(b) is the result of the simulation after 10 gradient steps. The goal of making the cubes stick together was accomplished. This experiment shows that physical states and control signals are interoperable between our differentiable framework and non-differentiable simulators.

2.8 Conclusion

We developed a scalable differentiable simulation method that has linear complexity with respect to the number of objects and collisions, and constant complexity with respect to spatial resolution. We use meshes as our core representation, enabling support for objects of arbitrary shape at any scale. A unified local collision handling scheme enables fast coupling between

objects with different materials. An acceleration scheme speeds up backpropagation by implicit differentiation. Experiments show that the presented techniques speed up backpropagation by large multiplicative factors, and their impact increases with the complexity of simulated scenes.

We demonstrated the application of the presented method in a number of case studies, including end-to-end gradient-based training of neural network controllers. Training with differentiable simulation was shown to be significantly more effective than gradient-free and model-free baselines.

One avenue for future work is to extend the presented framework to other types of objects, materials, and contact models. The framework is sufficiently general to support deformable solids and articulated bodies, as long as they have mesh-based representations with repulsive contact forces. We plan to extend our implementation to incorporate such object classes.

Chapter 3: Efficient Differentiable Simulation of Articulated Bodies

3.1 Introduction

Differentiable physics enables efficient gradient-based optimization with dynamical systems. It has achieved promising results in both simulated [4, 6] and real environments [7, 16]. Our goal is to make articulated body simulation efficiently differentiable. We aim to maximize efficiency in both computation and memory use, in order to support fast gradient-based optimization of differentiable systems that interact with articulated bodies in physical environments.

Articulated bodies play a central role in robotics, computer graphics, and embodied AI. Many control systems are optimized via experiences collected in simulation [122, 125, 126, 127]. However, they do not have access to analytic derivatives of the articulated dynamics. Therefore, nearly all gradient-based approaches have to design strategies to compute the gradients indirectly when dealing with articulated bodies.

The straightforward way to differentiate the simulation is to use existing automatic differentiation tools [25, 128, 129]. However, autodiff tools consume prohibitive amounts of memory when there are many simulation steps. Autodiff tracks every operation and stores the intermediate results in order to perform backpropagation. In articulated body simulation, the iterative contact solver and the dynamics algorithm [130] yield exceedingly long computational graphs. In our experiments, differentiable simulators built with autodiff tools run out of memory after 5,000

simulation steps – just a few seconds of experience. As a result, learning is constrained to short experiences or forced to use large time steps, thus curtailing the scope of behaviors that can be learned or undermining the simulation’s stability.

Furthermore, the overhead of creating and storing the auxiliary variables for autodiff also slows down the forward simulation. Although autodiff tools like DiffTaichi [19] and JAX [21] can accelerate the simulation of fluids and deformable solids by vectorization, it is difficult to achieve the same speedup in articulated body simulation because the articulated dynamics algorithm is highly serialized, unlike inherently parallel computation on grids and particles.

In this chapter, we design a differentiable articulated body simulation method that runs 10x faster with 1% of the memory consumption compared to differentiation based on autodiff tools. In order to minimize the overhead of differentiation, we derive the gradients of articulated body simulation using the adjoint method [131]. The adjoint method has been applied to fluids [31] and multi-body systems [35], but these applications do not support physically correct differentiable simulation of articulated bodies. Our derivation of the operator adjoints uses spatial algebra [132]. Our method needs almost no additional computation during the forward simulation and is an order of magnitude faster than autodiff tools.

We further reduce memory requirements by adapting ideas from checkpointing [27, 133] to the differentiable simulation setting. In the forward pass, we store the initial simulation state for each time step. During backpropagation, we recreate intermediate variables by reproducing simulation from the stored state. The overall runtime remains fast, while memory consumption is reduced by two orders of magnitude.

As an application of differentiable dynamics for articulated systems, we show that reinforcement learning (RL) can benefit from the knowledge of gradients in two ways. First, it can make

use of the gradients computed by the simulation to generate extra samples using first-order approximation. Second, during the policy learning phase, differentiable physics enables us to perform a one-step rollout of the objective value function so that the policy updates can be more accurate. Both schemes effectively improve the convergence speed and the attained reward. We also demonstrate applications of efficient differentiable articulated dynamics to inverse problems, such as motion control and parameter estimation. Gradient-based optimization enabled by our method accelerates convergence in these settings by more than an order of magnitude. Code is available on our project page: <https://github.com/YilingQiao/diffarticulated>

The contributions of this work are as follows:

- We derive the adjoint formulations for the entire articulated body simulation workflow, enabling a 10x acceleration over autodiff tools.
- We adapt the checkpointing method to the structure of articulated body simulation to reduce memory consumption by 100x, making stable collection of extended experiences feasible.
- We introduce two general schemes for accelerating reinforcement learning using differentiable physics.
- We demonstrate the utility of differentiable simulation of articulated bodies in motion control and parameter estimation, enhancing performance in these scenarios by more than an order of magnitude.

3.2 Related Work

Differentiable programming has been applied to rendering [86, 134, 135], image processing [136, 137], SLAM [138], and design [139]. Making complex systems differentiable enables learning

and optimization using gradient-based methods. Our literature review focuses on differentiable physics, the adjoint method, and neural approximations of physical systems.

Differentiable physics. Differentiable physics provides gradients for learning, control, and inverse problems that involve physical systems. Degraeve et al. [1] advocate using differentiable physics to solve control problems in robotics. de Avila Belbute-Peres et al. [2] obtain gradients of 2D rigid body dynamics. Liang et al. [3] use automatic differentiation tools to obtain gradients for cloth simulation. Qiao et al. [4] develop a more comprehensive differentiable physics engine for rigid bodies and cloth based on mesh representations. Ingraham et al. [5] present differentiable simulation of protein dynamics. For volumetric data, ChainQueen [6] computes gradients of the MPM simulation pipeline. Bern et al. [7] use FEM to model soft robots and perform trajectory optimization with analytic gradients. Takahashi et al. [8] differentiate the simulation of fluids with solid coupling.

Many exciting applications of differentiable physics have been explored [9, 10, 11, 12]. Huang et al. [13] propose a soft-body manipulation benchmark for differentiable physics. Toussaint et al. [14] manipulate tools with the help of differentiable physics. Song and Boularias [15] perform system identification by learning from the trajectory, and Song and Boularias [16] then use the estimated frictional properties to help robotic pushing.

A related line of work concerns the development of powerful automatic differentiation tools that can be used to differentiate simulation. DiffTaichi [19] provides a new programming language and a compiler, enabling the high-performance Taichi simulator to compute the gradients of the simulation. JAX MD [20] makes use of the JAX autodiff library [21] to differentiate molecular dynamics simulation. These works have specifically made use of vectorization on

both CPU and GPU to achieve high performance on grids and particle sets, where the simulation is intrinsically parallel. In contrast, the simulation of articulated bodies is far more serial, as dynamics propagates along kinematic paths rather than acting in parallel on grid cells or particles.

TinyDiffSim [22] provides a templated simulation framework that can leverage existing autodiff tools such as CppAD [23], Ceres [24], and PyTorch [25] to differentiate simulation. However, these methods introduce significant overhead in tracing the computation graph and accumulate substantial computational and memory costs when applied to articulated bodies.

Our method does not need to store the entire computation graph to compute the gradients. We store the initial states of each simulation step and reproduce the intermediate results when needed by the backward pass. This checkpointing strategy is also used in training neural ODEs [26] and large neural networks [27, 28, 29, 30]. We are the first to adapt this technique to articulated dynamics, achieving dramatic reductions in memory consumption and enabling stable simulation of long experiences.

Adjoint method. The adjoint method has been applied to fluid control [31], PDEs [32], light transport simulation [33], and neural ODEs [26, 34]. Recently, Geilinger et al. [35] proposed to use the adjoint method in multi-body dynamics. However, they operate in maximal coordinates and model body attachments using springs. This does not enforce physical validity of articulated bodies. In contrast, we operate in reduced coordinates and derive the adjoints for the articulated body algorithm and spatial algebra operators that properly model the body’s joints. This supports physically correct simulation with joint torques, limits, Coriolis forces, and proper transmission of internal forces between links.

Neural approximation. A number of works approximate physics simulation using neural networks [36,

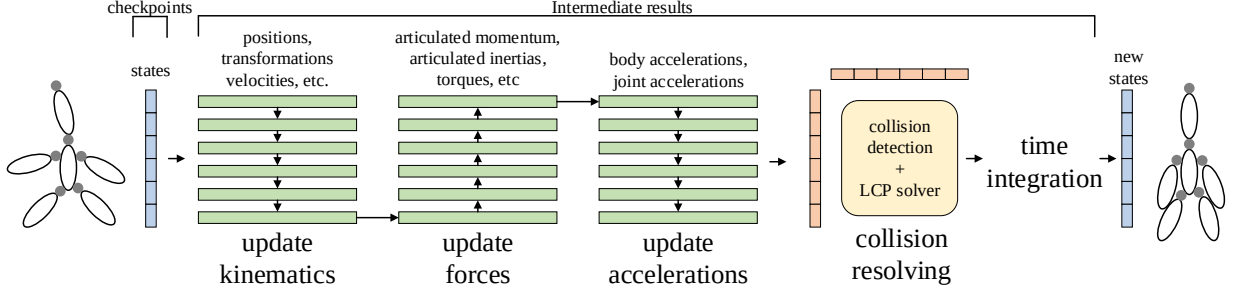


Figure 3.1: **The workflow of a simulation step.** Assume there are $n_q = 6$ DoF. The initial state is an $n_q \times 3$ matrix containing the position, velocity, and control input of each joint. The forward dynamics will traverse the articulated body sequentially three times. This process is difficult to parallelize and will generate a large number of intermediate results. After the forward dynamics there is a collision resolution step with collision detection and an iterative Gauss-Seidel solver. The size of the initial state (position, velocity, and control input) is much smaller than that of intermediate results but the initial state has all the information to resume all the intermediate variables.

[37, 38, 39, 40, 41, 42, 43, 44, 45, 46]. Physical principles have also been incorporated in the design of neural networks [47, 48, 49, 50, 51]. Approximate simulation by neural networks is naturally differentiable, but the networks are not constrained to abide by the underlying physical dynamics and simulation fidelity may degenerate outside the training distribution.

3.3 Preliminaries

Articulated body dynamics. For the forward simulation, we choose the recursive Articulated Body Algorithm (ABA) [130], which has $O(n)$ complexity and is widely used in articulated body simulators [122, 125, 126]. In each simulation step k , the states $\mathbf{x}^k$ of the system consist of configurations in generalized coordinates and their velocities $\mathbf{x}^k = [\mathbf{q}^k, \dot{\mathbf{q}}^k] \in \mathbb{R}^{2n_q \times 1}$, where n_q is the number of degrees of freedom in the system. Assuming each joint can be actuated by a torque, there is an n_q -dimensional control signal $\mathbf{u}^k \in \mathbb{R}^{n_q \times 1}$. The discretized dynamics at this step can be written as $\mathbf{f}^{k+1}(\mathbf{u}^k, \mathbf{x}^k, \mathbf{x}^{k+1}) = 0$.

Adjoint method. We can concatenate the states in the entire time sequence into $\mathbf{x} = [\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^{n_t}] \in \mathbb{R}^{n_t \cdot 2n_q \times 1}$,

with corresponding control input $\mathbf{u} = [\mathbf{u}^1, \mathbf{u}^2, \dots, \mathbf{u}^{n_t}] \in \mathbb{R}^{n_t \cdot n_u \times 1}$, where n_t is the number of

simulation steps. The concatenated dynamics equations can be similarly written as $\mathbf{f} = [\mathbf{f}^1, \mathbf{f}^2, \dots, \mathbf{f}^{n_t}] =$

0. For a learning or optimization problem, we would usually define a scalar objective function

$\Phi(\mathbf{x}, \mathbf{u})$. To get the derivative of this function w.r.t. the control input $\mathbf{u}$, one needs to compute

$$\frac{d\Phi}{d\mathbf{u}} = \frac{\partial\Phi}{\partial\mathbf{u}} + \frac{\partial\Phi}{\partial\mathbf{x}} \frac{d\mathbf{x}}{d\mathbf{u}}. \quad (3.3.0.1)$$

$\frac{\partial\Phi}{\partial\mathbf{u}}$ is easy to compute for each single simulation step. But it is prohibitively expensive to directly

solve $\frac{\partial\Phi}{\partial\mathbf{x}} \frac{d\mathbf{x}}{d\mathbf{u}}$ because $\frac{d\mathbf{x}}{d\mathbf{u}}$ will be a $2n_q n_t \times n_q n_t$ matrix.

Instead, we differentiate the dynamics $\mathbf{f}(\mathbf{x}, \mathbf{u}) = 0$ and obtain the constraint $\frac{\partial\mathbf{f}}{\partial\mathbf{x}} \frac{d\mathbf{x}}{d\mathbf{u}} = -\frac{\partial\mathbf{f}}{\partial\mathbf{u}}$.

Applying the adjoint method [131], $\frac{\partial\Phi}{\partial\mathbf{x}} \frac{d\mathbf{x}}{d\mathbf{u}}$ equals to

$$-\mathbf{R}^\top \frac{\partial\mathbf{f}}{\partial\mathbf{u}} \quad \text{such that} \quad \left(\frac{\partial\mathbf{f}}{\partial\mathbf{x}} \right)^\top \mathbf{R} = \left(\frac{\partial\Phi}{\partial\mathbf{x}} \right)^\top \quad (3.3.0.2)$$

Since the dynamics equation for one step only involves a small number of previous steps, the

sparsity of $\frac{\partial\mathbf{f}}{\partial\mathbf{u}}$ and $\frac{\partial\mathbf{f}}{\partial\mathbf{x}}$ makes it easier to solve for the variable $\mathbf{R}$ in Eq. 3.3.0.2.

$\frac{\partial\Phi}{\partial\mathbf{u}} - \mathbf{R}^\top \frac{\partial\mathbf{f}}{\partial\mathbf{u}}$ is called the adjoint of $\mathbf{u}$ and is equivalent to the gradient by our substitution. In

the following derivation, we denote the adjoint of a variable s by $\bar{s}$, and the adjoint of a function

$f(\cdot)$ by $\bar{f}(\cdot)$.

3.4 Efficient Differentiation

In this section, we introduce our algorithm for the gradient computation in articulated body simulation. For faster differentiation, the gradients of articulated body dynamics are computed by the adjoint method. We derive the adjoint of time integration, contact resolution, and forward dynamics in reverse order. We then adapt the checkpoint method to articulated body simulation to reduce memory consumption.

3.4.1 Adjoint Method for Articulated Dynamics

One forward simulation step can be split into five modules, as shown in Figure 3.1: perform the forward kinematics from the root link to the leaf links, update the forces from the leaves to the root, update the accelerations from the root to the leaves, detect and resolve collisions, and perform time integration. Backpropagation proceeds through these modules in reverse order.

Time integration. Backpropagation starts from the time integration. As an example, for a simulation sequence with $n_t = 3$ steps and an integrator with a temporal horizon of 2, the constraints $(\frac{\partial \mathbf{f}}{\partial \mathbf{x}})^\top \mathbf{R} = (\frac{\partial \Phi}{\partial \mathbf{x}})^\top$ in Equation 3.3.0.2 can be expanded as

$$\begin{bmatrix} (\frac{\partial \mathbf{f}^1}{\partial \mathbf{x}^1})^\top & (\frac{\partial \mathbf{f}^2}{\partial \mathbf{x}^1})^\top & 0 \\ 0 & (\frac{\partial \mathbf{f}^2}{\partial \mathbf{x}^2})^\top & (\frac{\partial \mathbf{f}^3}{\partial \mathbf{x}^2})^\top \\ 0 & 0 & (\frac{\partial \mathbf{f}^3}{\partial \mathbf{x}^3})^\top \end{bmatrix} \begin{bmatrix} \mathbf{R}^1 \\ \mathbf{R}^2 \\ \mathbf{R}^3 \end{bmatrix} = \begin{bmatrix} (\frac{\partial \Phi}{\partial \mathbf{x}^1})^\top \\ (\frac{\partial \Phi}{\partial \mathbf{x}^2})^\top \\ (\frac{\partial \Phi}{\partial \mathbf{x}^3})^\top \end{bmatrix} \quad (3.4.1.1)$$

If using the explicit Euler integrator, we have $\frac{\partial \mathbf{f}^k}{\partial \mathbf{x}^k} = \mathbf{I}$. Initially $\mathbf{R}^{n_t} = \frac{\partial \Phi}{\partial \mathbf{x}^{n_t}}$. The following $\mathbf{R}^k$

can be computed iteratively by

$$\mathbf{R}^k = \left(\frac{\partial \Phi}{\partial \mathbf{x}^k}\right)^\top - \left(\frac{\partial \mathbf{f}^{k+1}}{\partial \mathbf{x}^k}\right)^\top \mathbf{R}^{k+1}, k = 1, 2, \dots, n_t - 1. \quad (3.4.1.2)$$

When $\mathbf{R}^k$ backpropagates through time, the gradients $\overline{\mathbf{u}}^k$ can also be computed by Equation 3.3.0.2.

In fact, by the way we calculate $\mathbf{R}^k$, it equals the gradients of $\mathbf{x}^k$. Other parameters can also be computed in a similar way as $\overline{\mathbf{u}}^k$.

Collision resolution. The collision resolution step consists of collision detection and a collision solver. Upon receiving the gradients $\overline{\mathbf{x}}$ from the time integrator, the collision solver needs to pass the gradients to detection, and then to the forward dynamics. In our collision solver, we construct a Mixed Linear Complementarity Problem (MLCP) [140]:

$$\begin{aligned} \mathbf{a} &= \mathbf{A}\mathbf{x} + \mathbf{b} \\ \text{s.t. } \quad & \mathbf{0} \leq \mathbf{a} \perp \mathbf{x} \geq \mathbf{0} \text{ and } \mathbf{c}_- \leq \mathbf{x} \leq \mathbf{c}_+, \end{aligned} \quad (3.4.1.3)$$

where $\mathbf{x}$ is the new collision-free state, $\mathbf{A}$ is the inertial matrix, $\mathbf{b}$ contains the relative velocities, and $\mathbf{c}_-, \mathbf{c}_+$ are the lower bound and upper bound constraints, respectively. We use the projected Gauss-Seidel (PGS) method to solve this MLCP. This iterative solve trades off accuracy for speed such that the constraints $\mathbf{a}^\top \mathbf{x} = 0$ might not hold on termination. In this setting, where the solution is not guaranteed to satisfy constraints, implicit differentiation [2, 3, 4] no longer works. Instead, we design a reverse version of the PGS solver using the adjoint method to compute the gradients. Further details are in the supplement. In essence, the solver mirrors PGS, substituting the adjoints for the original operators. This step passes the gradients from the collision-free states to the forward dynamics.

Forward dynamics. Our articulated body simulator propagates the forward dynamics as shown in the green blocks of Figure 3.1. Each operation in the forward simulation has its corresponding adjoint operation in the backward pass. To compute the gradients, we derive adjoint rules for the operators in spatial algebra [132].

As a simple example, a spatial vector $\mathbf{p}_i = [\mathbf{w}, \mathbf{v}] \in \mathbb{R}^6$ representing the bias force of the i -th link is the sum of an external force $[\mathbf{f}_1, \mathbf{f}_2] \in \mathbb{R}^6$ and a cross product of two spatial motion vectors $[\mathbf{w}_1, \mathbf{v}_1], [\mathbf{w}_2, \mathbf{v}_2] \in \mathbb{R}^6$:

$$\begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 + \mathbf{w}_1 \times \mathbf{w}_2 + \mathbf{v}_1 \times \mathbf{v}_2 \\ \mathbf{f}_2 + \mathbf{w}_1 \times \mathbf{v}_2 \end{bmatrix}. \quad (3.4.1.4)$$

Once we get the adjoint $[\bar{\mathbf{w}}, \bar{\mathbf{v}}]$ of $\mathbf{p}_i = [\mathbf{w}, \mathbf{v}]$, we can propagate it to its inputs:

$$\begin{aligned} \begin{bmatrix} \bar{\mathbf{w}}_1 \\ \bar{\mathbf{v}}_1 \end{bmatrix} &= \begin{bmatrix} -\bar{\mathbf{w}} \times \mathbf{w}_2 - \bar{\mathbf{v}} \times \mathbf{v}_2 \\ -\bar{\mathbf{w}} \times \mathbf{v}_2 \end{bmatrix} \\ \begin{bmatrix} \bar{\mathbf{w}}_2 \\ \bar{\mathbf{v}}_2 \end{bmatrix} &= \begin{bmatrix} \bar{\mathbf{w}} \times \mathbf{w}_1 \\ \bar{\mathbf{w}} \times \mathbf{v}_1 + \bar{\mathbf{v}} \times \mathbf{w}_1 \end{bmatrix}, \quad \begin{bmatrix} \bar{\mathbf{f}}_1 \\ \bar{\mathbf{f}}_2 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{w}} \\ \bar{\mathbf{v}} \end{bmatrix} \end{aligned} \quad (3.4.1.5)$$

This example shows the adjoint of one forward operation. The time and space complexity of the original operation and its adjoint are the same as the forward simulation. The supplement provides more details on the adjoint operations.

3.4.2 Checkpointing for Articulated Dynamics

The input and output variables of one simulation step have relatively small dimensionalities (positions, velocities, and accelerations), but many more intermediate values are computed during the process. Although this is not a bottleneck for forward simulation because the memory allocation for intermediate results can be reused across time steps, it becomes a problem for differentiable simulation, which needs to store the entire computation graph for the backward pass. This causes memory consumption to explode when simulation proceeds over many time steps, as is the case when small time steps are needed for accuracy and stability, and when the effects of actions take time to become apparent.

The need for scaling to larger simulation steps motivates our adaptation of the checkpointing method [27, 133]. Instead of saving the entire computation graph, we choose to store only the initial states in each time step. We use this granularity because (a) the states of each step have the smallest size among all essential information sufficient to replicate the simulation step, (b) finer checkpoints are not useful because at least one step of intermediate results needs to be stored in order to do the simulation, and (c) sparser checkpoints will use less memory but require multiple steps for reconstructing intermediate variables, costing more memory and computation. We validate our checkpointing policy with an ablation study in the supplement. Figure 3.2 illustrates the scheme. During forward simulation, we store the simulation state in the beginning of each time step. During backpropagation, we reload the checkpoints (blue arrows) and redo the forward (one-step) simulation to generate the computation graph, and then compute the gradients using the adjoint method in reverse order (red arrows).

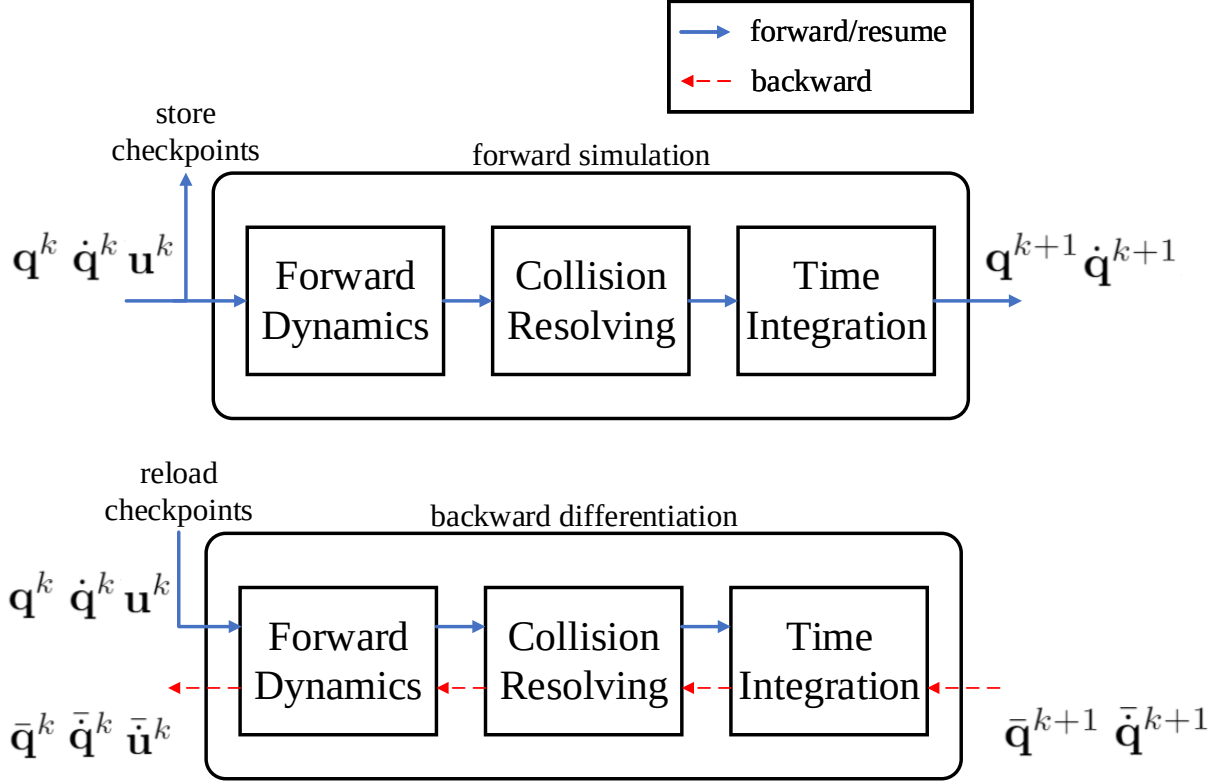


Figure 3.2: **Differentiation of articulated body dynamics.** *Top:* forward simulation at step k . The simulator stores the states $\mathbf{q}^k$, $\dot{\mathbf{q}}^k$, and control input $\mathbf{u}^k$. *Bottom:* During backpropagation, the checkpoints are reloaded and the simulator runs one forward step to reconstruct the intermediate variables. Beginning with the gradients from step $k+1$, we use the adjoint method to sequentially compute the gradients of all variables through time integration, the constraint solver, and forward dynamics.

In summary, assume the simulation step consists of two parts:

$$\mathbf{Z}^k = G(\mathbf{x}^{k-1}), \quad \mathbf{x}^k = F(\mathbf{Z}^k), \quad (3.4.2.1)$$

where $\mathbf{Z}^k$ represents all the intermediate results. After each step, we free the space consumed by $\mathbf{Z}^k$, only storing $\mathbf{x}^k$. During backpropagation, we recompute $\mathbf{Z}^k$ from $\mathbf{x}^{k-1}$ and use the adjoint method to compute gradients of the previous step:

$$\mathbf{Z}^k = G(\mathbf{x}^{k-1}), \quad (3.4.2.2)$$

$$\overline{\mathbf{Z}^k} = \overline{F}(\mathbf{Z}^k, \overline{\mathbf{x}^k}), \quad (3.4.2.3)$$

$$\overline{\mathbf{x}_{k-1}} = \overline{G}(\mathbf{x}_{k-1}, \overline{\mathbf{Z}^k}). \quad (3.4.2.4)$$

3.5 Reinforcement Learning

Our method can be applied to simulation-based reinforcement learning (RL) tasks to improve policy performance as well as convergence speed. By computing gradients with respect to the actions, differentiable simulation can provide more information about the environment. We suggest two approaches to integrating differentiable physics into RL.

Sample enhancement. We can make use of the computed gradients to generate samples in the neighborhood around an existing sample. Specifically, given a sample (s, a_0, s'_0, r_0) from the history, with observation s , action a_0 , next-step observation s'_0 , and reward r_0 , we can generate

new samples (s, a_k, s'_k, r_k) using first-order approximation:

$$\begin{aligned}\mathbf{a}_k &= \mathbf{a}_0 + \Delta\mathbf{a}_k, \\ \mathbf{s}'_k &= \mathbf{s}'_0 + \frac{\partial\mathbf{s}'_0}{\partial\mathbf{a}_0}\Delta\mathbf{a}_k, \\ r_k &= r_0 + \frac{\partial r_0}{\partial\mathbf{a}_0}\Delta\mathbf{a}_k,\end{aligned}$$

where $\Delta\mathbf{a}_k$ is a random perturbation vector. This method, which we call *sample enhancement*, effectively generates as many approximately accurate samples as desired for learning purposes. By providing a sufficient number of generated samples around the neighborhood, the critic can have a better grasp of the function shape (patchwise rather than pointwise), and can thus learn and converge faster.

Policy enhancement. Alternatively, the policy update can be adjusted to a format compatible with differentiable physics, usually dependent on the specific RL algorithm. For example, in MBPO [141], the policy network is updated using the gradients of the critic:

$$\mathcal{L}_\mu = -Q(\mathbf{s}, \mu(\mathbf{s})) + Z, \quad (3.5.0.1)$$

where $\mathcal{L}_\mu$ is the loss for the policy network μ , Q is the value function for state-action pairs, and Z is the regularization term. To make use of the gradients, we can expand the Q function one step forward,

$$\frac{\partial Q(\mathbf{s}, \mathbf{a})}{\partial\mathbf{a}} = \frac{\partial r}{\partial\mathbf{a}} + \gamma \frac{\partial Q(\mathbf{s}', \mu(\mathbf{s}'))}{\partial\mathbf{s}'} \frac{\partial\mathbf{s}'}{\partial\mathbf{a}}, \quad (3.5.0.2)$$

steps	50.0	100.0	500.0	1000.0	5000.0
Ceres	100.0	200.0	1100.0	2700.0	23 600.0
CppAD	16.0	26.0	160.0	320.0	3100.0
JAX	200.0	200.0	400.0	700.0	3000.0
PyTorch	1200.0	2400.0	11 700.0	12 400.0	N/A
Ours	0.3	0.3	0.7	1.2	5.0

Table 3.1: **Peak memory use of different simulation frameworks.** The memory footprint (MB) of our framework is more than two orders of magnitude lower than autodiff methods. PyTorch crashes at 5,000 simulation steps. ADF fails to compute the gradients in a reasonable time (10 min) and is not included here for this reason.

and substitute it in Eq. 3.5.0.1:

$$\mathcal{L}'_{\mu} = -\frac{\partial Q(\mathbf{s}, \mathbf{a})}{\partial \mathbf{a}} \mu(\mathbf{s}) + Z. \quad (3.5.0.3)$$

Eqs. 3.5.0.1 and 3.5.0.3 yield the same gradients with respect to the action, which provide the gradients of the network parameters. This method, which we call *policy enhancement*, constructs the loss functions with embedded ground-truth gradients so that the policy updates are physics-aware and generally more accurate than merely looking at the Q function itself, thus achieving faster convergence and even potentially higher reward.

3.6 Experiments

For experiments, we will first scale the complexity of simulated scenes and compare the performance of our method with autodiff tools. Then we will integrate differentiable physics into reinforcement learning and use our method to learn control policies. Lastly, we will apply

steps	50.0	100.0	500.0	1000.0	5000.0
ADF	25.7	25.5	25.1	32.1	58.4
Ceres	27.2	27.5	27.2	34.0	58.2
CppAD	2.4	2.4	2.3	2.3	4.5
JAX	53.3	46.1	43.1	42.7	42.3
PyTorch	195.6	192.2	199.2	192.8	N/A
Ours	0.3	0.3	0.2	0.2	0.2

Table 3.2: **Simulation time for forward pass.** Ours is at least an order of magnitude faster than autodiff tools (in msec). PyTorch crashes at 5,000 simulation steps. CppAD is the fastest baseline.

differentiable articulated dynamics to solve motion control and parameter estimation problems.

3.6.1 Comparison with Autodiff Tools

Using existing autodiff tools is a convenient way to derive simulation gradients. However, these tools are not optimized for articulated dynamics, which adversely affects their computation and memory consumption in this domain. We compare our method with state-of-the-art autodiff tools, including CppAD [23], Ceres [24], PyTorch [25], autodiff [142] (referred to as ADF to avoid ambiguity), and JAX [21]. All our experiments are performed on a desktop with an Intel(R) Xeon(R) W-2123 CPU @ 3.60GHz with 32GB of memory.

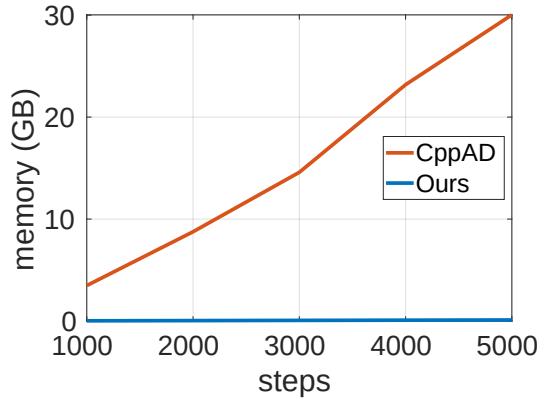
One robot. In the first round, we profile all the methods by simulating one Laikago robot standing on the ground. Figure 3.3(a) provides a visualization. The state vector of the Laikago has 37 dimensions (19-dimensional positions and 18-dimensional velocities). We vary the number of simulation steps from 50 to 5,000. JAX is based on Python and its JIT compiler cannot easily



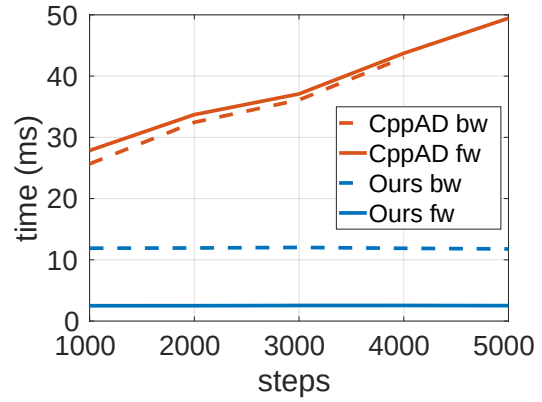
(a) One Laikago



(b) Ten Laikago robots



(c) Memory footprint



(d) Time per step

Figure 3.3: **Memory and speed of our method vs. CppAD**, the fastest autodiff method. (a,b) Two scenes used in experiments. (c,d) Memory consumption and per-step runtime when simulating ten Laikago robots for different numbers of steps. CppAD crashes when simulating for 5,000 steps due to memory overflow.

steps	50.0	100.0	500.0	1000.0	5000.0
Ceres	29.5	29.6	29.5	37.6	62.8
CppAD	2.4	2.3	2.3	2.4	4.8
JAX	148.1	125.7	127.6	129.6	126.6
PyTorch	275.7	273.3	280.8	272.1	N/A
Ours	1.2	1.1	1.2	1.1	1.2

Table 3.3: **Simulation time for the backward pass.** Ours is the fastest (in msec). PyTorch crashes at 5,000 simulation steps.

be used in our C++ simulation framework. To test the performance of JAX, we write a simple approximation of our simulator in Python, with similar computational complexity as ours. The simplified test code can be found in the supplement.

The memory consumption of different methods is listed in Table 3.1. The memory consumed by autodiff tools is orders of magnitude higher than ours. Among all the auto-differentiation tools, JAX and CppAD are the most memory-efficient. ADF fails to backpropagate the gradients in a reasonable time. We show in the supplement that the computation time of backpropagation in ADF is exponential in the depth of the computational graph. Note that articulated body simulation is intrinsically deep. In this experiment, the depth of one simulation step can reach 10^3 due to sequential iterative steps in the forward dynamics and the contact solver. PyTorch crashes at 5,000 simulation steps because of memory overflow.

Table 3.2 reports the average time for a forward simulation step. The numbers indicate that our method is faster than others by at least an order of magnitude. Our method has a constant time complexity per step. The time per step of ADF, Ceres, and CppAD increases with the number of steps. Note that JAX and Torch are optimized for heavily data-parallel workloads.

Articulated body simulation is much more serial in nature and cannot take full advantage of their vectorization. Table 3.3 reports the time for backward step. Our method is the fastest when computing the gradients.

Ten robots. To further test the efficiency of our method, we simulate a scene with 10 Laikago robots. Figure 3.3(b) provides a visualization. In this experiment, our method is compared with CppAD, the best-performing autodiff framework according to the experiments reported in Tables 3.1 and 3.2. The number of simulation steps is varied from 1,000 to 5,000. Figure 3.3(c) shows that the memory footprint of our method is negligible compared to CppAD. Our method only needs to store 5 KB of data per step, while CppAD needs to save the full data and topology of the computational graph. The forward and backward time per step are plotted in Figure 3.3(d). Our backward pass takes longer than our forward pass because the method first runs forward simulation to reconstruct the intermediate variables. Nevertheless, our method is much faster than CppAD and the performance gap grows with simulation length.

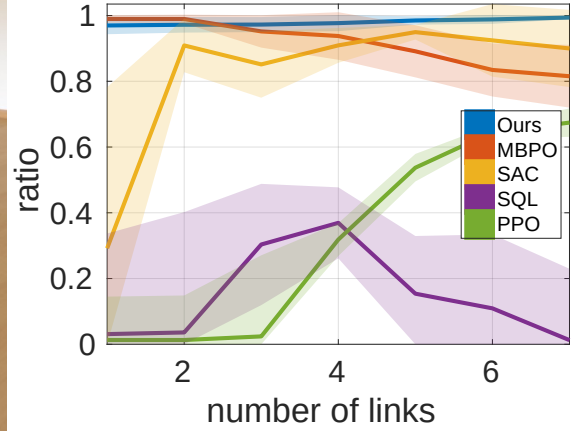
3.6.2 Integration with Reinforcement Learning

We demonstrate the improvements from the two RL enhancement methods described in Section 3.5. Note that we cannot use the two methods together because policy enhancement requires the gradients at the sample point, which the extra samples from sample enhancement do not have (unless higher-order gradients are computed). Thus, we test the two techniques separately. We use the model-based MBPO optimizer as the main baseline [141]. All networks use the PyTorch default initialization scheme.

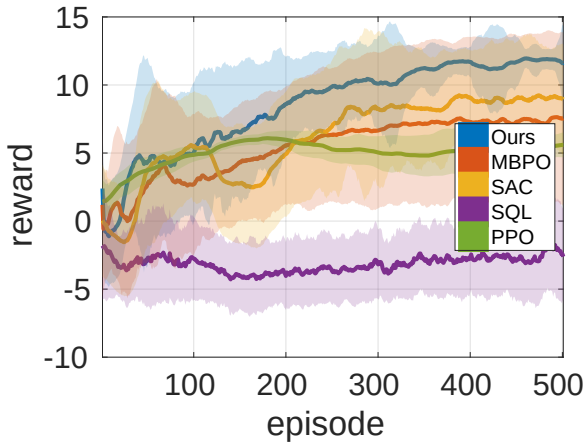
N-link pendulum. We first test our *policy enhancement* method in a simple scenario, where an



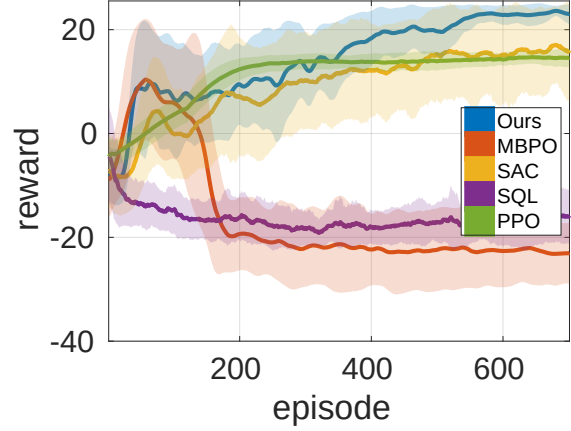
(a) Pendulum



(b) Relative reward



(c) 5-link learning curves



(d) 7-link learning curves

Figure 3.4: **The n -link pendulum task.** Our method attains higher reward than MBPO.

n -link pendulum needs to reach a target point by applying torques on each of the joints under gravity (Figure 3.4(a)). The target point is fixed to be the highest point reachable, and initially the pendulum is positioned horizontally. The reward function is the progress to the target between consecutive steps:

$$r = \|\mathbf{x}_t - \mathbf{x}_g\|^2 - \|\mathbf{x}_{t-1} - \mathbf{x}_g\|^2, \quad (3.6.2.1)$$

where $\mathbf{x}_t$ is the end-effector location at time t , and $\mathbf{x}_g$ is the target location.

We trained each model for $100n$ epochs, where n is the number of links. The results are shown in Figure 3.4. In Figure 3.4(b), we report the relative reward of each task, which is defined as the attained reward divided by the maximal possible reward. MBPO works well in easy scenarios with up to 3 links, but its performance degrades starting at 4 links. In contrast, our model reaches close to the best possible reward for all systems. The learning curves for the 5-link and 7-link systems are shown in Figure 3.4(c,d). MBPO does not attain satisfactory results for 6- and 7-link systems. We observed that around the 100th epoch, the losses for the Q function of the MBPO method increased a lot. We hypothesize that this is because the complexity of the physical system exceeds the expressive power of the model network in MBPO.

MuJoCo Ant. Next, we test our *sample enhancement* method on the MuJoCo Ant. In this scenario, a four-legged robot on the ground needs to learn to walk towards a fixed heading (Figure 3.5(a)). The scenario is the same as the standard task defined in MuJoCo, except that the simulator is replaced by ours. We also compare with other methods (SAC [143], SQL [144], and PPO [145] implemented in Ray RLlib [146]) for reference. For every true sample we get from the simulator, we generate 9 extra samples around it using our sample enhancement method. Other

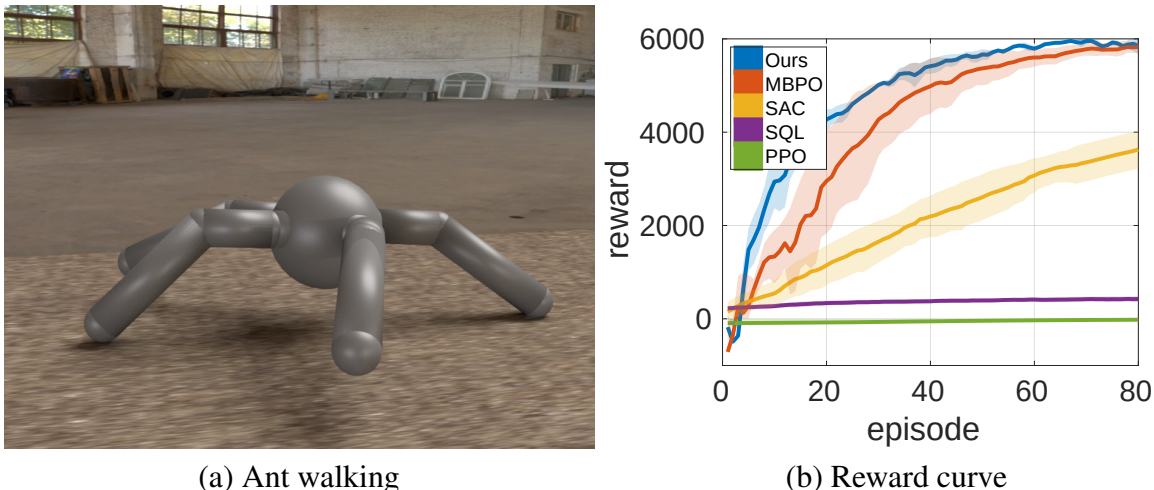


Figure 3.5: **The MuJoCo Ant task.** Using our differentiable simulator to generate extra samples accelerates learning.

than that, everything is the same as in MBPO. We repeat the training 4 times for all methods.

Figure 3.5(b) shows the reward over time. Our method exhibits faster convergence.

We can also transfer the policy trained in our differentiable simulator to MuJoCo. This tests the fidelity of our simulator and robustness of the learned policy to simulator details. The results are reported in the supplement.

3.6.3 Applications

Motion control. Differentiable physics enables application of gradient-based optimization to learning and control problems that involve physical systems. In Figure 3.6, we show two physical tasks: (a) throwing a ball to a target location using a 9-DoF robot arm and (b) hitting a golf ball to a target location using an articulated linkage. These tasks are also shown in the supplementary video.

For both scenarios, a torque vector is applied to the joints in each time step. Assume there are n time steps and k DoFs. The torque variable to be optimized has $n \times k$ dimensions,

and the objective function is the L2 distance from the ball’s actual position to the target. The joint positions and the torques are initialized as 0. If no gradients are available, a derivative-free optimization algorithm such as CMA-ES [123] can be used. We plot the loss curves of our method and CMA-ES in Figure 3.6.

In (a), SGD with gradients from our simulator converges in 20 steps. In contrast, CMA-ES does not reach the same accuracy even after 500 steps. In (b), SGD with gradients from our simulator converges in 50 steps, while CMA-ES does not reach the same accuracy even after 1500 steps.

Parameter estimation. Our method can estimate unknown physical parameters from observation. In Figure 3.7, a racecar starts with horizontal velocity $1m/s$. The wheels and the steering system are articulated. We estimate the sliding friction coefficient μ between the wheels and the ground such that the car stops at $x = 0.8m$ at the end of the episode. At the initial guess $\mu = 0.002$, the car reaches $x = 1.0m$. We use gradient descent to optimize the estimated friction coefficient. After 10 iterations of SGD with gradients provided by our method, the car reaches the goal with $\mu = 0.21$. In comparison, the gradient-free baseline takes multiple times longer to reach comparable objective values.

3.7 Conclusion

We have developed a differentiable simulator for articulated body dynamics that runs 10x faster with 100x smaller memory footprint in comparison to existing autodiff tools. To achieve this performance gain, we analyze the workload of each simulation step to better manage computation and memory. The adjoint method is used to compute the gradients of the simulation. We derive

the adjoints of spatial algebra and the Gauss-Seidel solver. We then adapt the checkpointing method to deal with the sequential nature of articulated body simulation, reducing memory consumption by two orders of magnitude.

As a supporting contribution, we have explored two applications of differentiable physics to reinforcement learning with physical systems, reporting preliminary results that indicate that differentiable physics can accelerate learning. We have also presented application scenarios that clearly demonstrate the effectiveness of gradient-based optimization in motion control and parameter estimation with articulated physical systems.

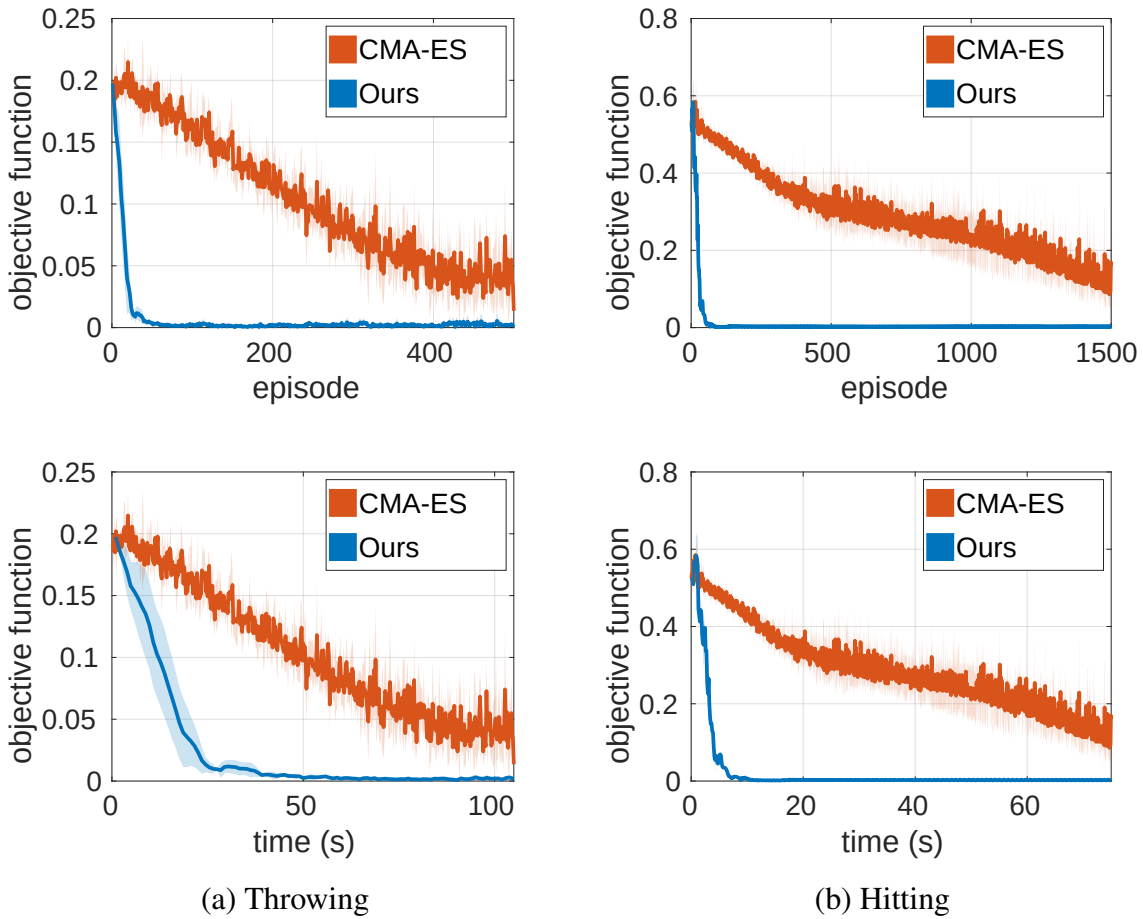
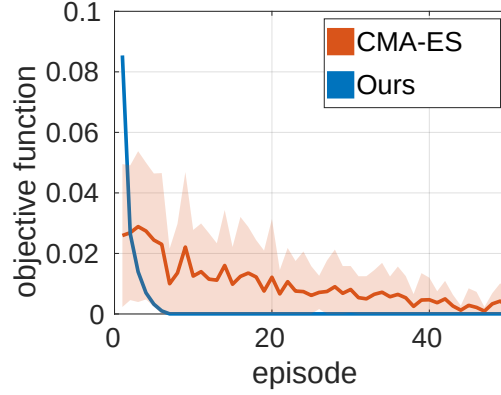


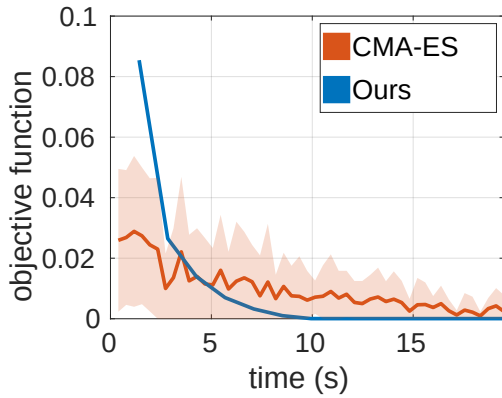
Figure 3.6: **Motion control.** The task is to optimize the torque vectors to (a) throw a ball with an articulated 9-DoF robot arm and (b) hit a golf ball with an articulated linkage. In both scenarios, the goal is for the ball to settle at a specified target location. SGD with gradients provided by our method converges within 20 and 50 steps, respectively, while gradient-free optimization fails to reach comparable accuracy even with more than an order of magnitude higher number of iterations. The third row indicates that even though CMA-ES does not compute gradients, our method is still faster overall in wall-clock time.



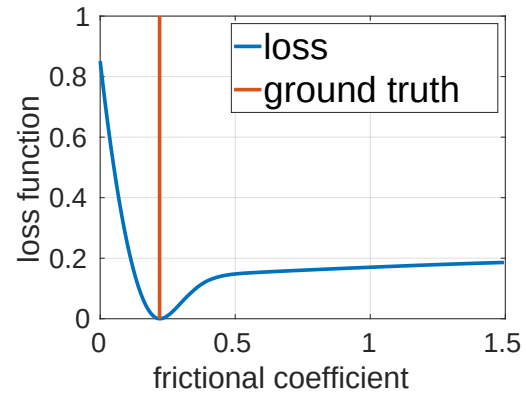
(a) A racecar



(b) Loss per episode



(c) Loss over time



(d) Loss landscape

Figure 3.7: Parameter estimation. The goal is to estimate the sliding friction coefficient that makes the racecar decelerate to a target location. (b,c) Loss curves in episode and wall-clock time, respectively. (d) Loss landscape plot. SGD with gradients computed by our method solves the task within 10 iterations, more accurately and faster than the gradient-free baseline.

Chapter 4: Differentiable Simulation of Soft Multi-body Systems

Soft articulated bodies have been studied and utilized in a number of important applications, such as microsurgery [52], underwater robots [147], and adaptive soft grippers [53]. Since the compliance of deformable materials can enable robots to operate more robustly and adaptively, soft biomimetic robots are drawing a lot of attention and have made considerable progress. A snailfish robot dives at a depth of 10,900 meters in the Mariana Trench [148]. Drones equipped with soft manipulators grasp and transmit objects with a 91.7% success rate [149]. Soft hands with pneumatic actuators are able to grasp objects of different shapes, including water bottles, eyeglasses, and sheets of cloth [150]. To enable rapid prototyping of soft robots and efficient design of control algorithms through virtual experiments, we aim to create a realistic deformable multi-body dynamics framework, in which soft articulated robots can be simulated to learn powerful control policies.

Design and control of soft robots are challenging because of their nonlinear dynamics and many degrees of freedom. Differentiable physics has shown great promise to deal with such complex problems [7, 13, 16, 139]. One possibility is to treat soft bodies as volumes that are modeled as sets of particles or finite elements [6, 151]. These methods have made great progress, but the volumetric representations are difficult to scale to large multi-body systems and are poorly suited to modeling internal skeletons. Moreover, contact handling in recent differentiable physics

frameworks [3, 19] often does not comply with Coulomb’s Law, which is central to plausible visual realism and correct physical behavior.

In this chapter, we design a powerful and accurate differentiable simulator for soft multi-body dynamics. Since our entire framework is differentiable, our method can be embedded with gradient-based optimization and learning algorithms, supporting gradient-based system identification, motion planning, and motor control. Within the simulator, we first use tetrahedral meshes to enable *adaptive resolution* and more accurate modeling. Next, to couple soft materials with articulated skeletons, we design a *top-down matrix assembly algorithm* within the local steps of Projective Dynamics [74]. For accurate contact handling, we extend and generalize a dry friction model previously developed for cloth simulation [78] to soft solids and introduce a new *matrix splitting strategy* to stabilize the solver. In addition, our simulation framework incorporates actuator models widely used in robotics, including muscles [152], joint torques [153], and pneumatic actuators [150]. With the support of the articulated skeleton constraints, dry frictional contact, and versatile actuators, our novel differentiable algorithm can simulate soft articulated robots and compute gradients for a wide range of applications.

The key contributions of this work are as follows.

- A top-down matrix assembly algorithm within Projective Dynamics to make soft-body dynamics compatible with reduced-coordinate articulated systems.
- An extended and generalized dry friction model for soft solids with a new matrix splitting strategy to stabilize the solver.
- Analytical models of muscles, joint torques, and pneumatic actuators to enable more realistic and stable simulation results.

- A unified differentiable framework that incorporates skeletons, contact, and actuators to enable gradient computation for learning and optimization.
- Experimental validation demonstrating that differentiable physics accelerates system identification and motion control with soft articulated bodies up to *orders of magnitude*.

In the following paper, Related Work section will discuss related papers in deformable body simulation and differentiable physics. Method Section will introduce a preliminary of projective dynamics to explain the high-level simulation framework and define notations that will be used later. Then, its following sections will be about how to deal with articulated skeleton and contact in our method. Experiments will show the ablation studies and comparisons results with other learning methods. Code is available on our project page: https://github.com/YilingQiao/diff_fem

4.1 Related Work

Deformable body simulation using Finite Element Method (FEM) plays an important role in many scientific and engineering problems [52, 53, 54]. Previous works model soft bodies using different representations and methods for specific tasks. There are several kinds of approaches for modeling body actuation. Pneumatic-based methods [55] change the rest shape to produce reaction forces. Rigid bones attached within soft materials are also used to control the motion of deformable bodies [56, 57, 58, 59]. To further simulate biologically realistic motion, it is common to apply joint torques in articulated skeletons [60]. For example, [61, 62] use articulated body dynamics to govern the motion while handling collisions using soft contact. Inspired by animals, different designs of muscle-like actuators for soft-body simulations were also proposed [63, 64,

65].

Regarding contact modeling, spring-based penalty forces are widely used [66, 67, 68] for their simplicity. More advanced algorithms include inelastic projection [69, 70] and barrier-based repulsion [71, 72]. However, these methods do not always conform to Coulomb’s frictional law. We opt for a more realistic dry frictional model [73] to better handle collisions.

Projective Dynamics [74] is widely used for its robustness and efficiency for implicit time integration. It has been extended to model muscles [75], rigid skeletons [76], realistic materials [77], and accurate contact forces [78]. Our method also adopts this framework for faster and more stable time integration. In contrast to the aforementioned methods using Projective Dynamics, our algorithm is the first to enable joint actuation in articulated skeletons together with a generalized dry frictional contact for soft body dynamics.

Differentiable physics has recently been successfully applied to solve control and optimization problems. There are several types of physically-based simulations that are differentiable, including rigid bodies [1, 2], soft bodies [6, 12, 19, 35, 151], cloth [3, 4], articulated bodies [154, 155, 156], and fluids [8, 32, 45, 157]. Differentiable physics simulation can be used for system identification [16, 158], control [9], and design [139, 159]. For differentiable soft-body dynamics, Du et al. [151] propose a system for FEM simulation represented by volume mesh. This system has been applied to robot design [160] and control [161]. Different from this work [151], our approach uses tetrahedral meshes with adaptive resolution to model finer detail and scale better to complex articulated bodies.

Krishna Murthy et al. [12], Hu et al. [19] use source code transformation to differentiate the dynamics, but their contact model does not follow Coulomb’s law. Geilinger et al. [35] simulate soft materials attached to rigid parts with penalty-based contact force, but their use of maximal

coordinates makes it difficult to incorporate joint torques. In comparison, our model has realistic contact handling, versatile actuators, and skeletons with joint constraints, thereby enabling our method to simulate a much wider range of soft, multi-body systems not possible before.

There are other works that approximate physical dynamics using neural networks [42, 43, 44, 162]. These methods are inherently differentiable but cannot guarantee physical correctness outside the training distribution.

4.2 Soft Body Simulation Using Projective Dynamics

We use Projective Dynamics [74] to model the physics of soft, multi-body systems because its efficient implicit time integration can make the simulation more stable. We briefly introduce Projective Dynamics below. The dynamics model can be written as

$$\mathbf{M}(\mathbf{q}_{n+1} - \mathbf{q}_n - h\mathbf{v}_n) = h^2(\nabla E(\mathbf{q}_{n+1}) + \mathbf{f}_{ext}), \quad (4.2.0.1)$$

with $\mathbf{M}$ being the mass matrix, $\mathbf{q}_n$ the vertex locations at frame n , h the time step, $\mathbf{v}_n$ the velocity, E the potential energy due to deformation, and $\mathbf{f}_{ext}$ the external forces. We choose implicit Euler for a stable time integration, then the state $\mathbf{q}_{n+1}$ can be solved by

$$\mathbf{q}_{n+1} = \arg \min_{\mathbf{q}} \frac{1}{2h^2}(\mathbf{q} - \mathbf{s}_n)^\top \mathbf{M}(\mathbf{q} - \mathbf{s}_n) + E(\mathbf{q}), \quad (4.2.0.2)$$

where $\mathbf{s}_n = \mathbf{q}_n + h\mathbf{v}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}$. Projective Dynamics reduces the computational cost by introducing an auxiliary variable $\mathbf{p}$ to represent the internal energy as the Euclidean distance

between $\mathbf{p}$ and $\mathbf{q}$ after a projection $\mathbf{G}$:

$$E(\mathbf{q}) = \sum_i \frac{\omega_i}{2} \|\mathbf{G}_i \mathbf{q} - \mathbf{p}_i\|_F^2, \quad (4.2.0.3)$$

where ω is a scalar weight, and E contains internal energy from different sources, such as deformations, actuators, and constraints. The computation of ω , $\mathbf{G}$, and $\mathbf{p}$ is dependent on the form of the energy. Combining all energy components into Eq. 4.2.0.2, we have

$$\mathbf{q}_{n+1} = \arg \min_{\mathbf{q}} \frac{1}{2} \mathbf{q}^\top \left(\frac{\mathbf{M}}{h^2} + \mathbf{L} \right) \mathbf{q} + \mathbf{q}^\top \left(\frac{\mathbf{M}}{h^2} \mathbf{s}_n + \mathbf{J} \mathbf{p} \right), \quad (4.2.0.4)$$

where $\mathbf{L} = \sum \omega_i \mathbf{G}_i^\top \mathbf{G}_i$ and $\mathbf{J} = \sum \omega_i \mathbf{G}_i^\top \mathbf{S}_i$, and $\mathbf{S}_i$ is the selector matrix. Since this is a quadratic optimization without constraints, its optimal point is given by the solution of the following linear system:

$$\left(\frac{\mathbf{M}}{h^2} + \mathbf{L} \right) \mathbf{q}_{n+1} = \frac{\mathbf{M}}{h^2} \mathbf{s}_n + \mathbf{J} \mathbf{p} \quad (4.2.0.5)$$

Note that the estimation of $\mathbf{p}$ is based on the current values of $\mathbf{q}$. Therefore we need to alternate between computing $\mathbf{p}$ and solving $\mathbf{q}$ until convergence. Luckily, both steps are fast and easy to solve: solving $\mathbf{q}$ in Eq. 4.2.0.5 is easy because it is a simple linear system, and computing $\mathbf{p}$ can be fast because it is local and can be parallelized. We show the generic Projective Dynamics method in Alg. 1.

Algorithm 1 Soft body simulation using Projective Dynamics

- 1: $\mathbf{x}^1 \leftarrow$ initial condition
 - 2: **for** $t = 1$ **to** $n - 1$ **do**
 - 3: **while** not converged **do**
 - 4: Compute $\mathbf{p}_i$ for all energy components i according to $\mathbf{q}$ (Local step)
 - 5: Solve $\mathbf{q}$ in Eq. 4.2.0.5 according to $\mathbf{p}_i$ (Global step)
-

4.3 Articulated Skeletons

Skeletons are indispensable for vertebrate animals and nowadays articulated robots. However, adding skeletons into the soft body simulation is challenging. First, the rigid bones cannot be simply replaced by soft materials with large stiffness, since this can make the system unstable and unrealistic. Second, joint connections between bones must be physically valid at all times, and thus also cannot be modeled as soft constraints. Moreover, the formulation should support joint actuation as torques to drive the multi-body system like an articulated robot. Li et al. [59] proposed a method for passive articulated soft-body simulation with ball joint constraints. We extend this method to enable rotational/prismatic joints, torque actuation, and precise joint connections without introducing extra constraints.

4.3.1 Rigid Body System

When integrated with hard skeletons, vertices on the rigid parts can be expressed as

$$\mathbf{q}_k = \mathbf{Q}\mathbf{T}_k^r\mathbf{V}_k, \quad (4.3.1.1)$$

where $\mathbf{Q} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix}$ is the projection from homogeneous coordinates to 3D coordinates, $\mathbf{T}_k^r \in \mathbb{R}^{4 \times 4}$ the rigid transformation matrix, and $\mathbf{V}_k \in \mathbb{R}^{4 \times m_k}$ the rest-pose homogeneous coordinates of the k^{th} rigid body.

During the global step in Projective Dynamics, we do not directly solve for $\mathbf{T}_k^r$, but for the **increment** $\Delta \mathbf{z}_k$ in its degree-of-freedom (DoF), to avoid nonlinearity. This formulation restricts

the changes of the rigid vertices to the tangent space yielded by the current $\mathbf{T}_k^r$:

$$\mathbf{q}_k^{i+1} = \mathbf{q}_k^i + \Delta \mathbf{q}_k^i \approx \mathbf{q}_k^i + \frac{\partial \mathbf{q}_k^i}{\partial \mathbf{z}_k} \Delta \mathbf{z}_k^i, \quad (4.3.1.2)$$

where $\mathbf{q}_k^i$ are the vertex locations of the k^{th} body in the i^{th} iteration step, and $\mathbf{z}_k$ is the variable defining the DoF of the k^{th} rigid body, including the rotation and the translation. The nonrigid part of the vertices can also be integrated with this formulation simply with $\frac{\partial \mathbf{q}_k^i}{\partial \mathbf{z}} = \mathbf{I}$.

Let $\mathbf{B} = \frac{\partial \mathbf{q}_k^i}{\partial \mathbf{z}}$ be the Jacobian of the concatenated variables. Eq. 4.2.0.4 can be rewritten as

$$\Delta \mathbf{z}^i = \arg \min_{\Delta \mathbf{z}} \frac{1}{2} \Delta \mathbf{z}^\top \mathbf{B}^\top \left(\frac{\mathbf{M}}{h^2} + \mathbf{L} \right) \mathbf{B} \Delta \mathbf{z} + \Delta \mathbf{z}^\top \mathbf{B}^\top \left(\left(\frac{\mathbf{M}}{h^2} + \mathbf{L} \right) \mathbf{q}^i - \left(\frac{\mathbf{M}}{h^2} \mathbf{s}_n + \mathbf{J} \mathbf{p} \right) \right) \quad (4.3.1.3)$$

After solving $\Delta \mathbf{z}^i$, the new vertex states $\mathbf{q}^{i+1}$ are derived by the new rigid transformation matrix $\mathbf{T}_k^{r'}$ using Eq. 4.3.1.1, which is subsequently computed in the local step, discussed below.

Local step. The variable $\Delta \mathbf{z}_k^i$ is composed of the increment of rotation ω_k^i and translation l_k^i based on the current transformation $\mathbf{T}_k^r$:

$$\Delta \mathbf{q}_k^i = \begin{bmatrix} -[\mathbf{q}_k^i]_\times & \mathbf{I} \end{bmatrix} \begin{bmatrix} \omega_k^i \\ l_k^i \end{bmatrix}. \quad (4.3.1.4)$$

Here $[\mathbf{q}_k^i]_\times$ is defined as the vertical stack of the cross product matrices of all vertices in the k^{th} rigid body. During the local step, we compute the SVD of the new transformation matrix after

integrating ω_k^i and l_k^i ,

$$\mathbf{T}_k = \begin{bmatrix} \mathbf{I} + \omega_k^{i*} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{T}_k^r + \begin{bmatrix} \mathbf{0} & l_k^i \\ \mathbf{0} & 0 \end{bmatrix} = \mathbf{U}\Sigma\mathbf{V}^\top, \quad (4.3.1.5)$$

and restrict it to $\text{SO}(3)$ to obtain the new rigid transformation, $\mathbf{T}_k^{r'} = \mathbf{U}\mathbf{V}^\top$.

The local step of Projective Dynamics for a single rigid body is the same as in [76]. However, we propose Eq. 4.3.1.2 to generalize the coupling to kinematic trees [163] with precise and actuated articulation.

4.3.2 Top-down Matrix Assembly for Articulated Body Systems

The articulated body system formulation is similar to the rigid body one, except that the transformation matrix is now chained:

$$\mathbf{T}_k^r = \prod_{u \in U_k} \mathbf{A}_u, \quad (4.3.2.1)$$

where U_k contains all ancestor of the k^{th} link (inclusive), and $\mathbf{A}_u$ is the local transformation matrix defined by joint u .

For rigid bodies, the vertex locations of a rigid body only depend on the body's own DoF variables. In the articulated system, however, they are also affected by the body's ancestors. Therefore, $\mathbf{B}$ changes from a block diagonal matrix to a block lower triangular matrix if the rigid body vertices are ordered by their kinematic tree depth.

To compute the matrix $\mathbf{B}$, we consider a link u with one of its non-root ancestor v . By the

definitions in Sec. 4.3.1, the corresponding block in matrix $\mathbf{B}$ is

$$\mathbf{B}_{u,v} = \frac{\partial \mathbf{T}_u^r \mathbf{V}_u}{\partial \mathbf{z}_v} = \mathbf{Q} \mathbf{P}_v \frac{\partial \mathbf{A}_v}{\partial \mathbf{z}_v} \mathbf{S}_{v,u} \mathbf{V}_u, \quad (4.3.2.2)$$

where $\mathbf{P}_v$ is the prefix product of the local transformation matrices of the link chain from root to v (exclusive), and $\mathbf{S}_{v,u}$ is the suffix product from v to u . In boundary cases where $u = v$, the formulation becomes

$$\mathbf{B}_{u,u} = \mathbf{Q} \mathbf{P}_u \frac{\partial \mathbf{A}_u}{\partial \mathbf{z}_u} \mathbf{V}_u. \quad (4.3.2.3)$$

When v is the root and thus has the same DoFs as a rigid body, using the results from Sec. 4.3.1, the formulation can be simplified to

$$\mathbf{B}_{u,root} = \mathbf{Q} \begin{bmatrix} -[\mathbf{q}_u]_{\times} & \mathbf{I} \end{bmatrix}. \quad (4.3.2.4)$$

Computing Eq. 4.3.2.2 requires the matrix products $\mathbf{P}_v$ and $\mathbf{S}_{v,u}$ of a link chain (v, u) in the tree. Straightforward approaches here could result in $O(N^3)$ complexity, where N is the number of links. However, by utilizing the kinematic tree and conducting the computation in top-down order, the complexity can be reduced to $O(N^2)$, which is optimal. The key observation is that the prefix and suffix products can be computed recursively:

$$\mathbf{P}_v = \mathbf{P}_{v'} \mathbf{A}_{v'} \quad (4.3.2.5)$$

$$\mathbf{S}_{v',u} = \mathbf{A}_v \mathbf{S}_{v,u}, \quad (4.3.2.6)$$

assuming v' is the parent link of v . When we traverse the kinematic tree in a depth-first order, the prefix product can be computed in $O(1)$. The suffix product is also obtained as we iterate along the path back to the root. Algorithm 2 shows the matrix assembly method starting from the root node:

Algorithm 2 Matrix Assembly for the Articulated System

- 1: Input: tree link u
 - 2: Compute $\mathbf{P}_u$ using Eq. 4.3.2.5
 - 3: $v \leftarrow u$
 - 4: **while** v is not root **do**
 - 5: Compute $\mathbf{S}_{v,u}$ using Eq. 4.3.2.6
 - 6: Compute $\mathbf{B}_{u,v}$ using Eq. 4.3.2.2
 - 7: $v \leftarrow \text{parent}(v)$
 - 8: Compute $\mathbf{B}_{u,\text{root}}$ using Eq. 4.3.2.4
 - 9: **for** s **in** descendants(u) **do**
 - 10: Solve link s recursively
-

The transformation matrix $\mathbf{A}$ and the Jacobian of a joint depend on the joint type. This is further derived in Appendix C.

4.3.3 Articulated Joint Actuation

Eq. 4.3.1.3 is a quadratic optimization, so the optimal $\Delta \mathbf{z}^i$ is given by the linear system

$$\mathbf{H} \Delta \mathbf{z}^i = \mathbf{k}, \quad (4.3.3.1)$$

where $\mathbf{H} = \mathbf{B}^\top \left(\frac{\mathbf{M}}{h^2} + \mathbf{L} \right) \mathbf{B}$ and $\mathbf{k} = -\mathbf{B}^\top \left(\left(\frac{\mathbf{M}}{h^2} + \mathbf{L} \right) \mathbf{q}^i - \left(\frac{\mathbf{M}}{h^2} \mathbf{s}_n + \mathbf{Jp} \right) \right)$. Reordering the vertices into sets of deformable and rigid ones yields the following partitioning of the matrix:

$$\begin{bmatrix} \mathbf{H}_d & \mathbf{H}_c^\top \\ \mathbf{H}_c & \mathbf{H}_r \end{bmatrix} \begin{bmatrix} \Delta \mathbf{z}_d^i \\ \Delta \mathbf{z}_r^i \end{bmatrix} = \begin{bmatrix} \mathbf{k}_d \\ \mathbf{k}_r \end{bmatrix}, \quad (4.3.3.2)$$

where $*_d$ and $*_r$ represents the deformable and the rigid parts, respectively. The joint actuation can be directly added to $\mathbf{k}_r$ since the linear system is analogous to the basic formulation $\mathbf{Ma} = \mathbf{f}$ where the right hand side represents the sum of forces and/or torques. The formulation of pneumatic and muscle actuators can be found in Appendix C.

4.4 Contact Modeling

We handle the contact using Coulomb's frictional law via a Jacobian. To compute the velocities after collisions, we split the left-hand side of Equation 4.2.0.5 into the diagonal mass matrix $\mathbf{M}$ and the constraint matrix $h^2\mathbf{L}$, and move the latter to the right-hand side:

$$\mathbf{M}\mathbf{v}^{i+1} = \mathbf{f} - h^2\mathbf{L}\mathbf{v}^i + \xi^i, \quad (4.4.0.1)$$

where $\mathbf{f} = \mathbf{M}\mathbf{s}_n - (\mathbf{M} + h^2\mathbf{L})\mathbf{q}_n + h^2\mathbf{Jp}$ and the contact force ξ^i is determined according to $\mathbf{f} - h^2\mathbf{L}\mathbf{v}^i$ (the current momentum) to enforce non-penetration and static/sliding friction. The idea here is to enforce Coulomb's law at every iteration, which is ensured by solving $\mathbf{v}^{i+1}$ using the inverse of a diagonal matrix $\mathbf{M}$. As long as the solver converges at the end, the final $\mathbf{v}$ and ξ will conform to the frictional law.

This method works well for cloth contacts [78], but cannot be directly applied to soft

bodies, because solid continuum is much stiffer than thin sheets, i.e. the elements in $h^2\mathbf{L}$ on the right-hand side are much larger than those in $\mathbf{M}$ on the left-hand side, resulting in severe oscillation or even divergence during the iterative solve.

We show that in order to guarantee the convergence of Equation 4.4.0.1, the time step h has to satisfy a certain condition:

Proposition 4.4.1. *Assuming $\mathbf{f}$ and ξ are fixed, Equation 4.4.0.1 converges if the time step h satisfies*

$$h^2 < \frac{\rho}{24\sqrt{3}T\mu \sum_{k=1}^3 \|\mathbf{q}_k - \mathbf{q}_0\|_2^2} \quad (4.4.0.2)$$

where ρ is the density, μ is the stiffness, T is the number of tetrahedra, and $\mathbf{q}_i$ are the vertex positions.

Details of the proof can be found in Appendix C. Using the setting in our experiments, where $T \approx 1000$, $\mu \approx 3 \times 10^5$, $\|\mathbf{q}_k - \mathbf{q}_0\|_2 \approx 10^{-2}$, and $\rho \approx 1$, we would need to set $h < 1/1934$ in order to ensure the convergence, which is too strict for the simulation to be useful in general applications.

Splitting scheme. We improve this method to be compatible with soft body dynamics by introducing a new splitting scheme. Eq. 4.4.0.1 is reformulated as

$$(\mathbf{M} + h^2\mathbf{D})\mathbf{v}^{i+1} = \mathbf{f} - h^2(\mathbf{L} - \mathbf{D})\mathbf{v}^i + \xi^i, \quad (4.4.0.3)$$

where $\mathbf{D}$ are the diagonals of $\mathbf{L}$. Our key observation is that (a) the diagonals of $\mathbf{L}$ are necessary and sufficient to stabilize the Jacobian iteration, and (b) adding extra diagonal elements to the

left-hand side will not break the Coulomb friction law. We show in Appendix C that under the same assumption as Proposition 4.4.1, our method is guaranteed to converge no matter how big h is. This improvement accelerates the simulation since larger time steps mean faster computation.

We also note that the new splitting scheme will not modify the behavior of the collision response because the convergence point of Eq. 4.4.0.1 is the same as that of Eq. 4.4.0.3, and thus Coulomb’s Law is still satisfied at convergence.

4.5 Experiments

In this section, we first introduce our implementation and then report ablation studies that demonstrate the importance of skeletons and collision contacts in soft-body dynamics. Subsequently, we use the gradients computed by our method to perform system identification; specifically, we estimate the physical parameters of bridges. Finally, we perform gradient-based learning of grasping and motion planning on robots with various actuators, including a pneumatic gripper, an octopus with muscles, and a skeleton-driven fish. Our method can converge more than an order of magnitude faster than reinforcement learning and derivative-free baselines.

4.5.1 Implementation

Our simulator is written in C++, the learning algorithms are implemented in PyTorch [25], and Pybind [164] is used as the interface. We run our experiments on two desktops, one with an Intel Xeon W-2123 CPU @ 3.6GHz and the other with an Intel i9-10980XE @ 3.0GHz, respectively. For differentiation, the numerical data structure in our simulator is templated and integrated into the C++ Eigen library, such that our method can conveniently interoperate with

autodiff tools to differentiate the dynamics. Our method can also run in pure C++ to perform forward simulation. We refer to the open-source code from [75] (Apache-2.0), [78] (GNU GPL v3.0), and [59] (MPL2) during our implementation. More details can be found in our code in the supplement.

To further improve the memory efficiency, we introduce a checkpointing scheme [27] into our pipeline. Instead of storing the entire simulation history, we only store the system’s state in each step. During the backward pass, we reload the saved state vector and resume all the intermediate variables before the backpropagation. This strategy can save a major part of the memory, compared to the brute-force implementation. We conduct an experiment to compare

Table 4.1: Memory footprint (GB).

steps	w/o ckpt	w/ ckpt
10	0.9	0.1
20	1.4	0.1
100	6.9	0.1
200	15.7	0.1

the memory consumption with and without this checkpointing scheme. The results are reported in Table 4.1. CppAD [23] is used to differentiate the simulation here. In this experiment, we simulate a bridge and estimate its material properties as shown in Figure 4.3(a). The results show that the memory footprint of the baseline scales linearly with simulation length, while our checkpointing scheme keeps memory consumption nearly constant.



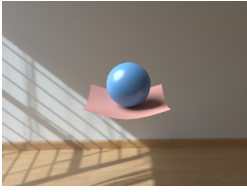
	Bone length error	Body deformation error	Joint angle error	Runtime (ms)
No skeleton	0.13 ± 0.02	0.20 ± 0.14	2.45 ± 0.87	132 ± 2
Rigid [4]	0	0.07 ± 0	1.81 ± 0	727 ± 258
Passive [59]	0	0.04 ± 0.03	2.34 ± 0.78	163 ± 4
MPM [19]	0.48 ± 0.32	0.12 ± 0.03	NaN	11 ± 0
Ours	0	0.05 ± 0.03	0.41 ± 0.30	191 ± 4

Figure 4.1: Ablation study of skeleton realization. ‘Bone length error’ measures the length change of a rigid bone, ‘body deformation error’ measures the deviation from the desired body length, and ‘joint angle error’ is the deviation from the target joint configuration. For all metrics, lower is better and 0 in error indicates the highest accuracy possible. Our model is the most physically realistic.

4.5.2 Ablation Study

Skeleton constraints. Controlling soft characters via skeletons is natural and convenient: vertebrate animals are soft, but are driven by piecewise-rigid skeletons. Our simulator supports skeletons and joint torques within soft bodies. This ablation study compares other designs with ours. In this experiment, a Baymax model [165] in its T-pose is released from above the ground, as shown in Figure 4.1. We embed 5 bones inside Baymax (4 in arms and legs, and 1 in the torso). When Baymax falls to the ground, we also add torques on its shoulders so it can lift its arms to a target Y-pose. More details of the setting and qualitative results can be found in Appendix C and the supplementary video. Three metrics, summarized in Figure 4.1, are used to measure realism. The metrics are averaged over 5 repetitions with different initial positions and velocities. For comparison, we simulate a ‘No skeleton’ Baymax without the support of rigid bones. Its bone error is non-zero because of the deformation. The Baymax in a differentiable rigid body simulator [4] is rigid, so the body length error is non-zero. Li et al. [59] simulate the ‘Passive’ skeleton case where there is no joint actuation and joint angles cannot be adjusted to the desired configuration. We also run DiffTaichi-MPM [19] by converting the mesh model to the point-based MPM representation. ‘MPM’ does not have skeletons so the errors are high. The arms also detach from the body so the joint error is NaN. Our method attains the highest degree of physical realism and correctness overall.

Contact handling. Good contact handling is critical for simulating multi-body systems that interact with their environment. In this experiment, we throw a 3D soft ball against a 2D thin sheet. Metrics in this experiment are penetration error and indicators of vertical compression and horizontal stretching. Zero penetration error is ideal. ‘Yes’ for compression/stretching indicates



	Penetration error	Compression	Stretching	Runtime (ms)
Cloth [78]	0.041 ± 0.004	No	No	930 ± 52
Rigid [4]	0	No	No	520 ± 135
MPM [19]	NaN	Yes	Yes	14 ± 0
Ours	0	Yes	Yes	51 ± 3

Figure 4.2: Ablation study of collision handling scheme. ‘Penetration error’ is greater than 0 if objects interpenetrate as a result of the simulation. ‘Compression’ and ‘stretching’ indicate whether the simulation allows the ball to compress and stretch in the vertical and horizontal directions, respectively. Our method can model a soft ball correctly while preventing interpenetration.

that the simulator can model the deformation of the soft ball correctly. The metrics are averaged over 5 experiments with different initial positions and velocities. The dry frictional contact model of Ly et al. [78] does not model the deformation of soft solids, and there could be penetration when the resolutions of the ball and cloth differ a lot due to the nodal collision handling scheme. The rigid differentiable simulator of Qiao et al. [4] can prevent interpenetration, but the ball remains rigid. MPM [19] can model the deformation of both the ball and the cloth, but the cloth is torn apart by the ball and penetration cannot be quantified. In contrast, our method accurately handles collision to avoid interpenetration and correctly simulates the deformation of the ball.

4.5.3 Applications

System identification. Determining the material parameters of deformable objects can be challenging given their high dimensionality and complex dynamics. In this experiment, we use our differentiable simulator to identify the material property of each finite element cell within the soft body. As shown in Figure 4.3, there are two bridges with unknown materials: a suspension bridge with both ends fixed and the entire bridge being soft, and an arch bridge that has three piers attached to the ground. Given that the movement of the barycenter under gravity, compared to its rest pose, is $\Delta x = 8cm$, we estimate Young’s modulus and Poisson’s ratio of each finite element cell in the

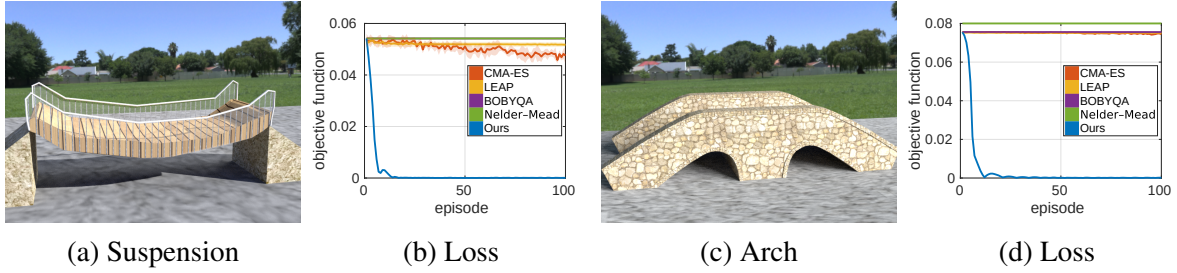


Figure 4.3: System identification. Given the desired displacement under gravity, we estimate Poisson’s ratio and Young’s modulus of each element of the two bridges with different structures. Our method converges much faster than gradient-free methods.

bridge. The loss function is the distance from the actual barycenter to the target. The suspension bridge has $n = 668$ cells and the arch one has $n = 2911$ cells. The number of unknowns is $2n$. We compare our method with four derivative-free methods (CMA-ES [123], LEAP [166], BOBYQA [167], and Nelder-Mead [168]). Each experiment is repeated 5 times with different random seeds. As shown in the figure, our method converges in ~ 10 iterations while others fail to converge even after 100 iterations, indicating that derivative-free methods in this high-dimensional setting become too inefficient to converge to a reasonable solution. By making use of the gradients provided by our method, common gradient-based algorithms can quickly reach the target configuration.

Motion planning. Controlling the motion of deformable bodies is challenging due to their flexible shapes. In this experiment, summarized in Figure 4.4, the task is to control robots with different actuator types. In general, given an initial state X_0 , control policy $\phi_{\theta_1}(\cdot)$, and material parameters θ_2 , the simulator can generate a trajectory of the states at all time steps t , $\{sim_t(X_0, \phi_{\theta_1}(\cdot), \theta_2)\} = \{X_1, X_2, \dots, X_n\}$. If we want the system to reach a target state X_{target} at the end of the simulation, we can define an objective function $L(X_0, \theta_1, \theta_2) = ||(sim_N(X_0, \phi_{\theta_1}(\cdot), \theta_2) - X_{target})||_2$, where the optimization variable are θ_1 and/or θ_2 .

We compare our method with Reinforcement Learning algorithms (SAC [143], SQL [144],

and PPO [145]), and the best derivative-free optimization method from the last experiment, CMA-ES. We also tried MBPO [169], but we found that this method takes too much memory and could not finish any test. All RL methods use the negative of the loss as the reward.

The **pneumatic gripper** in Figure 4.4(a) has 56 pneumatic cells in four arms and is attached to an (invisible) drone as in [149]. The pneumatic activation can control the volume of a tetrahedron. When the cells inflate, the arms will move inwards and hold the ball tighter. We control the pneumatic activation as well as the movement of the drone to move the ball from the start $(0, 0, 0)$ to our target $(0, 0.3, 0)$ in 50 steps. The loss is the distance from the actual position to the target position. Our method converges in 10 episodes while CMA-ES and PPO gradually converge in 200 and 500 episodes, respectively.

The **muscle-driven octopus** in Figure 4.4(b) has 8 legs, each with 2 muscles inside. It moves forward by actuating the muscles, being pushed by drag and thrust forces induced by the water on the octopus’s surface [75]. The octopus starts at $(0, 0, 0)$ and our target location is $(-0.4, 0.8, -0.4)$. We set the objective to be the distance between the current location and the desired location. The length of the simulation is 400 steps, and the control input in each step is 64-dimensional. In total, there are $64 \times 400 = 25600$ variables to optimize. Our method converges in 50 episodes while other methods fail to converge in 500 episodes.

The fish with an **embedded skeleton** in Figure 4.4(c) has 6 bones: 3 in its body, 2 in the fins, and 1 in the tail. The hydrodynamics in this environment is the same as in the octopus experiment. The fish starts at $(0, 0, 0)$ in step 1 and the target location in step 100 is $(0, 0, 0.15)$. The objective function is the distance from the actual location to the target location. For each step, there will be a torque vector of size 5 that represents the joint actuation level. In total, the optimization variable has 500 dimensions. Our method with gradient-based optimization can

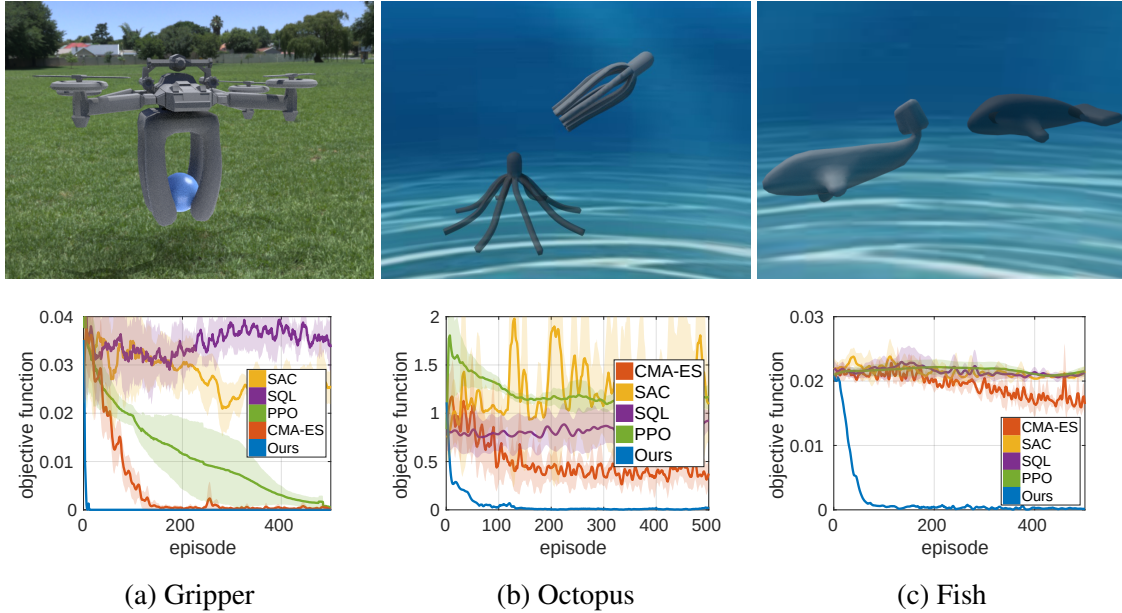


Figure 4.4: Motion control experiments. We simulate (a) a gripper with pneumatic cells in four arms, (b) an octopus with 16 muscles, and (c) a fish with 6 bones inside the body. The goal is to optimize the actuation schedules to get the robots to the target location. Our method is up to *orders of magnitude* faster than reinforcement learning and derivative-free baselines.

converge in roughly 50 episodes, while others cannot converge even after 500 episodes.

In summary, gradient-free optimization methods and RL algorithms meet substantial difficulties when tackling problems with high dimensionality, such as soft, multi-body systems. Even when the action space is as small as the one in the gripper case, RL methods still fail to rapidly optimize the policy. By introducing the gradients of the simulation, simple gradient-based optimization outperforms other algorithms. This work hopefully may inspire improvements in RL algorithms that tackle such high-dimensional problems.

4.6 Conclusion

This chapter has developed a differentiable physics framework for soft, articulated bodies with dry frictional contact. To make the simulation realistic and easy-to-use, we designed a recursive matrix assembly algorithm and a generalized dry frictional model for soft continuum

with a new matrix splitting strategy. Integrated with joint, muscle, and pneumatic actuators, our method can simulate a variety of soft robots. Using our differentiable physics to enable gradient-based optimization, our method converges more than an order of magnitude faster than the baselines and other existing alternatives.

There are some limitations in our contact handling and soft body dynamics. Currently, though our algorithm is more extensive and generalized than existing differentiable physics algorithms and our implementation handles the most commonly found contact configuration, vertex-face collisions, there could still be edge-edge penetration missed in some corner cases. Moreover, the Projective Dynamics pipeline limits the energy to have the form $E = \|\mathbf{G}\mathbf{q} - \mathbf{p}\|$. Some nonlinear material models (e.g., neo-Hookean) are not captured in this framework and new models for differentiable physics will be required for handling nonlinear and heterogeneous materials. For future work, we aim to add edge-edge collision handling in the Projective Dynamics pipeline. The techniques in [77] can be used to incorporate additional material types. GPU or other parallel computing implementation can be used to boost the performance of gradient computation.

Chapter 5: Other Differentiable Dynamical Systems

5.1 Traffic Simulation

Automobile traffic is a dynamic phenomenon that emerges from the interactions of a multitude of distinct entities, and its complexity has led analysts to rely on simulations to solve complex real-world traffic problems, such as signal control, congestion, road design, and urban planning [170]. Traffic simulation has become increasingly more important due to population growth, fast-growing technologies, and demands for autonomous driving. Autonomous driving agents need accurate and efficient models to simulate all types of plausible traffic scenarios to capture corner cases and accelerate learning-based training that requires data not easily available from real-world capturing [171].

Techniques for traffic simulation can be broadly classified as either *macroscopic* or *microscopic* based on their modeling assumptions. Macroscopic models describe traffic evolution as a system of partial differential equations (PDE), and the traffic state is represented as a *continuum* of values across the road network. Since we treat vehicles as particles translated by convection under this assumption, it is computationally efficient but provides coarse simulation results. In contrast, microscopic models represent traffic through individual vehicles, or agents, that are evolved individually and which together characterize the traffic state of the road network [172]. Therefore, this approach gives us fine-grained details but often requires much more computation



Figure 5.1: **Traffic simulation in an urban environment.** Traffic simulation can be used to analyze complex traffic dynamics. Among two popular traffic models, by simulating areas of interest with the microscopic one and using macroscopic simulation elsewhere, we can reduce the overall computational cost without compromising significant details. For instance, the center of intersection can be simulated with discrete, agent-based models for higher-fidelity inter-vehicle dynamic interaction. (*This scene is rendered in CARLA [173].*)

resources than macroscopic models.

In this work, we propose to adopt both microscopic and macroscopic models to create a more general traffic simulator that is differentiable (Figure 5.1). This kind of hybrid approach has already been proven to simulate large-scale traffic environments much more efficiently than either of the approaches while maintaining integrity [172]. Here we leverage the power of the hybrid approach to support large-scale traffic scenes, which will be demonstrated in our experiments. During simulation, our framework computes gradient information that provides abundant insight into traffic dynamics. This approach results in enhanced sample efficiency in our simulation, as each sample of traffic simulation comes with gradient information explaining how and why such an event has occurred. Moreover, this differentiability allows us to integrate our framework with neural networks to support a wide range of traffic control, planning, management, and flow optimization problems.

We first derive the analytical formulation of the differentiable traffic models for both macroscopic and microscopic paradigms. The resulting representations can simulate traffic on road networks for any metropolitan region more efficiently. Furthermore, we present a probabilistic method to

convert traffic states between macroscopic and microscopic ones, which is also differentiable. With this novel differentiable traffic simulation model, we can find near-optimal solutions to some traffic problems that were otherwise not previously solvable, and we can do so efficiently. To summarize, in this chapter we introduce the following main results:

- Derivation of a differentiable macroscopic traffic model based on ARZ model [174, 175];
- Analytical formulation of differentiable Intelligent Driver Model [176] for agent-based vehicle dynamics;
- Differentiable conversion between deterministic and probabilistic instantiation of two traffic models;
- Application of the first differentiable hybrid traffic simulation framework to traffic control .

In our experiments, we observe up to *an order of magnitude* speedup in runtime performance, in both forward simulation and back propagation process, compared to a baseline differentiable simulator based on automatic differentiation.

This is a collaboration work led by Sanghyun Son. More details can be found in the full paper [177].

5.2 Fluids Simulation

Differentiable physics has been introduced recently as a powerful and effective approach to solving control and inverse problems due to the differentiability that makes it possible to compute the gradient with user-defined objective functions in a neural network. The computed gradient allows us to use gradient-based optimizers, such as gradient descent and quasi-Newton method,

that are generally much more efficient than derivative-free optimization methods. Given the larger number of degrees of freedoms (DOFs) for complex dynamical systems (e.g., DOFs of fluids can be more than 10k) and their high computational cost for simulation over many time steps, it is indispensable to take advantage of the gradient information to efficiently minimize the objective functions.

Because of the effectiveness, many researchers investigated differentiable physics formulations and their applications, e.g., for rigid bodies [178, 179], articulated rigid bodies [35, 180], deformable solids [181], thin shells and cloths [3], coupling of rigid bodies and cloths [4], smokes [182, 183], and liquids [184]. While some differentiable formulations for fluid materials have been proposed, approaches to achieving differentiable physics differ from each other, typically due to differences in the underlying fluid simulation methodologies. In the literature, there are two types of commonly used fluid simulation methods characterized by distinct spatial discretization techniques: Eulerian grid-based approach and Lagrangian particle-based approach. While the Lagrangian particle-based approach has become popular due to its flexibility, it is known to be challenging to enforce the fluid incompressibility, which plays an important role for realistic fluid behaviors. By contrast, Eulerian grid-based approach has been proven to be efficient and effective in simulating incompressible fluids.

The differentiable approach for Eulerian fluid simulation has been proposed, e.g., with deep neural networks to predict fluid behaviors [182, 185, 186, 186]. However, it is reported that predicting the complex fluid behaviors for a long term can be difficult [185, 186]. In addition, fluids involving solid objects can pose additional challenges because of the increased possible fluid and solid configurations, which further make it difficult to learn and predict the fluid dynamics. As such, for reliable predictions of fluid behaviors involving solid interactions, it

is typically necessary to compute the fluid and solid dynamics using numerical simulation based on their governing equations.

In this chapter, we propose a new differentiable fluid simulation method that can be integrated as a part of layers in deep neural networks. The integrated differentiable simulator enables end-to-end learning of the neural networks with back propagation of the gradient through both of the simulator and networks. The trained networks can generate control inputs for the simulator to handle fluid and solid control problems. Our key contributions are summarized as follows.

1. Our differentiable fluid simulator offers the capability of handling one-way coupled solids due to a variational principle enforcing the fluid incompressibility for realistic fluid behaviors, while appropriately applying the free-slip boundary condition at the sub-grid, fluid-solid interface.
2. To achieve the differentiability, we apply the adjoint method to the entire simulation steps of dynamical systems (unlike partial application of the adjoint method within one time step, e.g., as in [3, 178, 182]), thereby enabling efficient and scalable gradient computations for user defined objective functions.
3. Our differentiable fluid simulator computes the gradient based on automatic differentiation (AD) at the high-level, fluid simulation operations (e.g., advection and pressure projection), avoiding the expensive reverse mode AD at the low-level, primitive operations (e.g., addition and subtractions of each variable), commonly implemented in public libraries.

Our validations and experiments demonstrate that our differentiable simulator can achieve at least one order of magnitude performance improvement, compared to the state-of-the-art techniques in inverse and control problems.

This is a collaboration work led by Tetsuya Takahashi. More details can be found in the full paper [8].

5.3 Analog Quantum Computing

Quantum computing has promised unprecedented improvement in our computational ability to tackle classically intractable problems. Despite of the rapid development of quantum hardware [187, 188], near-term quantum computers are still likely to have very restricted hardware resources, where the limited number of “qubits” and non-negligible machine noises would impede the implementation of large-scale quantum algorithms. Recent research results in both computer science [189] and physics [190] suggest a promising approach of designing resource-efficient *Noisy Intermediate-Scale Quantum (NISQ)* [191] applications by breaking quantum circuit abstractions and directly designing applications at the pulse-level control of quantum machines.¹ The benefits of this analog-oriented approach have been witnessed in the history of classical analog computing that predates digital computing due to relaxed hardware requirement and plays an important role in domain applications such as simulation.

One leading algorithmic paradigm on NISQ computers is the *Variational Quantum Algorithm (VQA)* with a few prominent examples like the Variational Quantum Eigensolver (VQE) [192], quantum approximate optimization algorithm (QAOA) [193], and more in [194]. Specifically, VQA uses *parametrized quantum models* to characterize loss functions, in particular those from quantum physics, which are potentially intractable for classical computing [195]. These parameters will then be optimized, usually through gradient-based approaches, to minimize the given loss function. Conventionally, parameterized quantum models are typically quantum circuits where

¹Pulse-level control is available on IBMQ computers.

each gate is parameterized by classical variables. Moreover, auto-differentiation techniques have been recently developed (e.g., [196, 197, 198]) for a scalable quantum gradient calculation on parameterized quantum circuits. We henceforth refer this existing framework as *differentiable digital quantum computing*.

Although parameterized quantum circuits are designed for NISQ applications, implementing (digital) quantum circuits still incurs non-negligible overheads, which significantly restricts the size of parametrized circuits that can be executed faithfully on NISQ machines. Moreover, the current parameterization in VQAs is also largely restricted by available parameterized gates and how they compose circuits, which in turn limits the expressive power of VQAs. A natural alternative to the current digital parameterization is to perform VQA directly on *parametrized analog signals (pulses)*, either on digital quantum machines with pulse-level controls, *or* on general analog quantum hardware. Indeed, parameterized analog pulses have the potential for more efficient NISQ implementation and better expressiveness as suggested in a recent perspective paper [190], which could be a more favorable parameterization for NISQ applications even when digital quantum gates are available.

However, there is not yet a systematic study of the analog parameterization for VQAs, its scalable training, and its quantitative benefits in achieving high-quality solutions in VQA applications.

Contributions. We conduct the first systematic study of the parameterization and scalable training of analog VQAs, which we call *differentiable analog quantum computing*. We also leverage our scalable

	Diff. Quantum	Neural ODE	Diff. Physics
$\frac{d}{dt}\mathbf{x}(t)=?$	$-iH(\mathbf{v}, t)\mathbf{x}(t)$	$f(\mathbf{x}(t), \mathbf{v})$	$f(\mathbf{x}(t), \mathbf{v})$
Parametrization	Basis Function	Neural Network	Dynamics Equation
Forward	Time Evolution	Forward Pass	Time Integration
Backward	MC Integration	Autodiff	Auto/Manual Diff.
Device	Quantum	Classical	Classical

Table 5.1: Machine learning in different dynamical systems.

training to demonstrate the quantitative benefits of the analog parameterization for specific VQA applications.

Specifically, we formulate the general differentiable analog quantum computing as a control problem, $\min_{\mathbf{v}} \mathcal{L}(\mathbf{v})$, where the loss function $\mathcal{L}(\mathbf{v})$ is calculated from the $\mathbf{v}$ -parametrized quantum state $\mathbf{x}(T; \mathbf{v})$ generated by quantum machines at the evolution time $t = T$. Intuitively, this control problem is no different from any classical ones except that the evolution of the quantum state in $[0, T]$ is governed by the Schrödinger equation

$$\frac{d}{dt}\mathbf{x}(t) = -iH(\mathbf{v}, t)\mathbf{x}(t), \quad (5.3.0.1)$$

where the Hamiltonian operator $H(\mathbf{v}, t)$ can be much more flexibly parameterized in $\mathbf{v}$ compared with parameterized quantum circuits. We also develop a scalable Monte Carlo integration technique of computing quantum-related gradients from the loss function $\mathcal{L}(\mathbf{v})$. A well-known difficulty in computing quantum-related gradients by classical means is the exponential cost associated with classical simulation of the quantum system. Thus, any scalable solution must leverage quantum machines to compute the gradients for themselves. Existing “auto-differentiation” techniques for parameterized quantum circuits (e.g., [196, 197, 198]) are designed for discrete-time evolution and the basic parameterized units (i.e., gates) are described by finite-dimensional matrices. Differential analog quantum computing exploits parameterization of continuous-time evolution and $H(\mathbf{v}, t)$ refers to a parameterized model from a functional space. Our Monte Carlo integration technique is designed to bridge this technical gap, which is later integrated with the stochastic gradient descent (SGD) for the entire framework for fast convergence and robustness

against empirical noise. We further illustrate that our quantum stochastic gradient descent could still work with approximate descriptions of $H(\mathbf{v}, t)$ for domain applications.

An analogy could be drawn between our approach and classical deep learning, as summarized in Table 5.1. *Neural ODEs* [34, 199] view the structure of ResNet [200], $\mathbf{x}_{n+1} = \mathbf{x}_n + f(\mathbf{x}_n, \theta)$, as the solving of an ordinary differential equation,

$$\frac{d}{dt}\mathbf{x}(t) = f(\mathbf{x}(t), \mathbf{v}), \quad (5.3.0.2)$$

with $f(\cdot)$ as the network layer, $\mathbf{v}$ the network parameter, and $\mathbf{x}(t)$ the hidden state. This formula is similar to our system in (5.3.0.1), although we adopt a different parametrization than neural networks. Similarly, *differentiable physics* (e.g., [181]), which incorporates physical simulation into learning process, leverages existing numerical solvers and the autodiff functionality of deep learning with backpropagation to compute gradients of a physical or dynamical system, then integrates them into a neural network. This approach has proven to accelerate learning and generalization.

Differentiable analog quantum computing can be deemed as a special form of differentiable physics at the quantum scale. Similar to the promise of differentiable physics or neural ODEs, we have also observed the advantageous performance of differentiable analog quantum computing against the conventional parameterized quantum circuits by orders of magnitude in major VQA applications like optimization and control. Our auto-differentiation technique in quantum is however less efficient than classical ones that leverage the chain rule and the forward/backward propagation. This is due to the quantum no-cloning theorem [201] which prevents us from storing any arbitrary immediate state during quantum computation. So, we develop a sampling technique

to compute gradients. To sum up, the key contributions of this work include:

- A formulation of differentiable, resource-efficient *analog* quantum computing framework,
- A scalable technique of computing gradients by Monte Carlo sampling with SGD,
- Formal analysis on correctness, efficiency, and robustness that showcases *exponential reduction of computational complexity* and bounded errors with approximate machine description,
- Applications of our framework on quantum optimization and control, with demonstrated advantages by *orders of magnitude* against parameterized quantum circuits. Our code is available here:

<https://github.com/YilingQiao/diffquantum>

This is a collaboration work. More details can be found in the full paper [202].

Part III

Reality-based Simulation

Chapter 6: Editable Neural Geometry and Physics from Monocular Videos

6.1 Introduction

Monocular RGB videos are ubiquitous today, and computers’ ability to understand spatial and physical properties embedded in their contents is essential for many applications. Can you imagine one day being able to build a fully-interactive digital twin of your surrounding world using just a smartphone? You could capture a moving, fully 3D portrait of yourself (Sec. 6.4.1), estimate the physical material parameters of objects and interact with them virtually (Sec. 6.4.3), playback existing recordings from new angles (Sec. 6.4.2), or even edit your favorite cartoon videos (Sec. 6.4.4). In this chapter, we design a system that makes progress toward realizing all of these things. Taking a single RGB video as input, our method estimates the geometry and dynamics parameters of the scene, parameterized by neural fields [79], and enables subsequent editing and physically-based simulation.

While humans can easily infer 3D shapes and physical properties of their environment even from a single eye, these tasks remain challenging for computer algorithms. Geometry reconstruction from a single view is an intrinsically under-constrained problem: there is usually more than one 3D structure that could result in a given 2D projection of a scene. Another key challenge lies in choosing a suitable representation to model the complex geometry of the real world. Explicit representations like surface mesh [203, 204] are widely used in computer

graphics, but they suffer from limited resolution and fixed topology. Implicit representations such as signed distance functions (SDFs) describe complex geometric structure more elegantly, but are not directly suitable for reasoning about dynamics properties. Estimating dynamics parameters is yet more challenging, since it is contingent upon correctly estimating geometry and its variation over time.

In this chapter, we design an end-to-end differentiable rendering and simulation pipeline that can directly learn the geometry and dynamics from a monocular video. By assuming the existence of a one-to-one dense mapping between temporal frames in 3D space, our method decouples the geometry estimation task into two complementary parts: a static reference frame, and a mapping function that encodes motion. The static reference frame consists of three neural fields: Appearance, Rigidity, and an SDF for geometry. Given a 3D spatial coordinate as input, these fields respectively output its RGB color, rigidity mask, and signed distance to the nearest surface, in the reference frame. Time-variant motion is encoded in a separate Motion field, which takes as input a 3D spatial coordinate (sampled along a ray), plus a learned latent code denoting a particular discrete time step, and outputs the 3D displacement from the sampled coordinate back to its corresponding reference frame coordinate. We utilize the density function from NeuS [88] for rendering, and the Rigidity mask from Non-Rigid NeRF [106] to regularize the motion and obtain foreground/background separation.

A convenient property that emerges from factorizing the scene into static geometry + dynamic motion fields is that by querying the motion field, we can establish a point-wise correspondence across frames, which we leverage to infer dynamics properties of the scene. Specifically, physics parameters are optimized by minimizing the difference between results from the differentiable simulator [151] and motion field. Upon learning reasonable parameters, the differentiable simulator

can be used for physically-based animation and editing of the scene. Experiments show that our method outperforms competing approaches in surface reconstruction, video reconstruction, and novel view synthesis tasks, both qualitatively and quantitatively. Multiple examples also demonstrate our full pipeline’s physically-based scene understanding and interactive scene-editing capabilities. Our code and data are available here: <https://sites.google.com/view/neuphysics>

In summary, the key contributions of this work are as follows:

- We present a framework for estimating and editing the geometry, appearance, and physics parameters directly from a single monocular video (Figure 6.3).
- By decoupling the learning of static SDF geometry and dynamic motion, our method effectively regularizes this under-constrained problem and enables *reconstruction of high-quality dynamic surface mesh sequences from monocular videos* (Figure 6.4).
- By connecting the neural-fields-based geometry reconstruction to a differentiable physics engine, this approach eliminates the need for manual construction of intermediate 3D models, and avoids poor initialization for the differentiable simulator, empowering users to *estimate the modeling and physics parameters from scratch* (Figure 2.9).
- We design a two-way conversion (Figure 6.2& 6.3) between the explicit hexahedral mesh and implicit neural fields in our end-to-end modeling, simulation and rendering pipeline, enabling users to (1) *extract high-quality explicit meshes* from the implicit fields, and (2) *interact with the implicit 3D field (e.g. add, delete, move, deform, and simulate)* by directly manipulating its explicit mesh counterpart.

6.2 Related Work

Differentiable volume rendering with implicit neural representations has recently been used to synthesize novel views, estimate geometry and appearance from images and videos. Neural Radiance Fields (NeRF) [79], and following works [80, 81, 82, 83] have achieved photorealistic rendering results even on complex scenes. The coordinate-based multi-layer perceptron (MLP) representation equips the scene with high resolution [84], can be effectively regularized [85], and is memory-efficient. Compared with differentiable surface rendering [86, 87], volume rendering can backpropagate gradients through the entire space, which leads to better optimization of complex geometry that contains topological changes, and mitigates getting stuck in local minima. The original density field from NeRF cannot directly reconstruct high-quality surface meshes, so neural SDF fields [88, 89, 90, 205, 206] have been proposed for multi-view geometry reconstruction. Compared to other discrete representations like meshes [91, 92, 93], point clouds [94, 95] and voxels [96, 97], SDF fields have higher resolution, and do not require pixel-level segmentation [98, 99, 100] for reconstruction tasks. Current approaches that employ neural SDF fields do not converge to accurate geometry when faced with dynamic scenes, while our method does so from a single video, which we show in figure 6.4.

Recent work has extended NeRF to videos of dynamic scenes [101, 102, 103], but they primarily focus on 2D image synthesis, not reconstructing 3D geometry. These works either directly condition the neural fields on time [108, 109, 110] or they learn a canonical field separately from time-dependent motion [104, 105, 106, 107]. We choose the latter strategy because it better constrains the learning problem based on a reasonable inductive bias, and importantly, it provides us with point-wise correspondence between time steps that is useful

for dynamics estimation. By representing geometry with a neural SDF field, and applying strong geometric priors, we can reconstruct dynamic 3D geometry better than existing dynamic extensions of NeRF, while even improving image reconstruction as shown in Section 6.4.1.

Differentiable simulators enable physics priors to better fit into data-driven approaches, and have therefore increasingly been applied to robotics and computer graphics tasks [9, 16, 139, 207, 208, 209]. These physically-based simulation methods have been developed for dynamical systems with different properties, including rigid bodies [1, 2, 4, 210, 211], soft bodies [6, 12, 19, 35, 151, 212, 213, 214], articulated bodies [154, 155, 156], cloth [3, 215], and fluids [8, 32, 45, 157].

6.3 Methods

Given a monocular video consisting of N RGB image frames $\{\mathbf{I}_i\}_{i=1}^N$, our approach estimates the 3D geometry within the camera’s field of view, point-wise correspondence of the dynamic 3D scene across time, and static/dynamic segmentation. By integrating the differentiable physics module, we can learn physics parameters from the dynamic motion sequence, which we leverage for video editing and forecasting. Figure 6.1 shows an overview of this chapter.

6.3.1 Rendering Pipeline

SDF and color networks. Similar to recent Neural Fields methods [79], we parameterize the geometry and appearance of the scene using coordinated-based Multi Layer Perceptrons (MLPs), which can render 2D images using the volume rendering scheme proposed in NeuS [88]. Geometry is represented by an SDF network $s_{\theta_{SDF}}(\mathbf{p}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ that maps a spatial coordinate

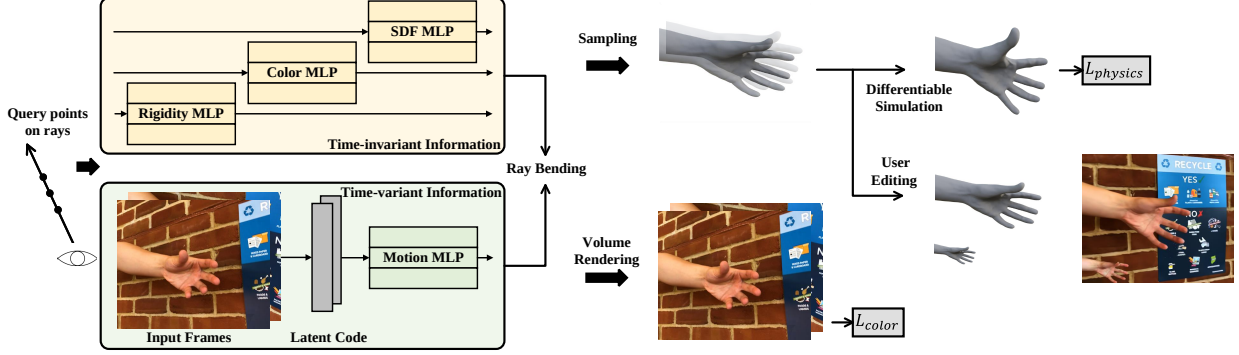


Figure 6.1: **Overall workflow.** *NeuPhysics* takes a monocular video as supervision. The time-invariant fields are defined on a reference frame, while time-dependent motion is applied to each point sampled in space. We then compute color loss from the volume rendered images. A differentiable simulator is embedded after optimizing the neural fields, to learn dynamics parameters and allow scene editing. A detailed computational graph can be found in Figure 6.3.

$\mathbf{p} \in \mathbb{R}^3$ to its signed distance to the nearest surface. Appearance is represented by another MLP $\mathbf{c}_{\theta_{color}}(\mathbf{p}) : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ that describes the RGB value of each spatial point. During rendering, the points along a ray cast from a given pixel can be written as $\{\mathbf{p}(t) = \mathbf{o} + t\mathbf{v} | t \geq 0\}$, where $\mathbf{o}$ is the camera center, $\mathbf{v}$ is the unit direction of the ray, and t is distance from the camera center. The pixel’s color is then calculated by integrating along the ray:

$$\mathbf{C}(\mathbf{o}, \mathbf{v}) = \int_0^{+\infty} w(s(\mathbf{p}(t))) \cdot \mathbf{c}(\mathbf{p}(t)) dt, \quad (6.3.1.1)$$

where $w(\cdot)$ [88] is a weight function describing the contribution of each point’s color based on its SDF value. More details can be found in the Appendix.

Motion network. However, the above framework is only designed to handle multi-view images of a static scene, which motivates another component to accommodate temporal information. Instead of directly conditioning the existing networks upon time, we choose to decouple the dynamic and static scene components. Therefore, the SDF and color networks are defined on a static reference frame, while a motion network $\mathbf{b}_{\theta_{motion}}(\mathbf{l}_i, \mathbf{p}) : \mathbb{R}^{d_l} \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$ estimates

the offset field [106] for each time step, where $\mathbf{l}_i \in \mathbb{R}^{d_l}$ is a learned latent feature encoding temporal information about the i -th frame $\mathbf{I}_i$. The motion network bends each query ray $\{\mathbf{p}(t)\}$ to $\{\mathbf{p}'(t) = \mathbf{p}(t) + \mathbf{b}(\mathbf{l}_i, \mathbf{p}(t))\}$, which is then queried on the reference frame for SDF and color values.

Rigidity network. Since we want to segment and simulate dynamic objects within the scene downstream, we apply a rigidity network [106] $r_{\theta_{\text{rigidity}}}(\mathbf{p}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ to distinguish dynamic foreground from static background. The mask $r(\mathbf{p}) \in [0, 1]$ defined on the reference frame indicates the likelihood that a point belongs to the swept volume of moving elements in the scene. The offset value becomes $\mathbf{b}_{\theta_{\text{motion}}}^{\theta_{\text{rigidity}}}(\mathbf{l}_i, \mathbf{p}(t)) = r_{\theta_{\text{rigidity}}}(\mathbf{p}) \cdot \mathbf{b}_{\theta_{\text{motion}}}(\mathbf{l}_i, \mathbf{p}(t))$. This formulation pushes dynamic regions toward larger $r(\cdot)$ values during training, while driving static areas toward zero.

6.3.2 Simulation Pipeline

Geometry reconstruction. Given the signed distance field $s(\cdot)$ and rigidity mask $r(\cdot)$, we can reconstruct the geometry of the dynamic and static parts of the scene. For example, the foreground in the i -th frame is contained within the swept volume of moving objects described by:

$$\mathcal{A}_i = \{\mathbf{p} | s(\mathbf{p} + \mathbf{b}(\mathbf{l}_i, \mathbf{p})) \leq k_{\text{sdf}}, r(\mathbf{p} + \mathbf{b}(\mathbf{l}_i, \mathbf{p})) \geq k_{\text{rigidity}}\}, \quad (6.3.2.1)$$

where k_{sdf} and k_{rigidity} are constant thresholds for the SDF and rigidity values, respectively.

This formulation is amenable to conversion into an explicit mesh representation. For rendering purposes, a *surface mesh* could be obtained using the marching cubes algorithm [216]. Alternatively,

we represent the dynamic elements as a *volume mesh*. We sample over a discrete 3D grid to find vertices within volume $\mathcal{A}_i$, and if a grid point $\mathbf{p}$ satisfies Equation 6.3.2.1, we add that voxel to a hexahedral volume mesh $\mathbf{m}_i$. We choose hexahedral rather than tetrahedral volume mesh due to its simplicity and regular structure, and because hexahedral mesh can be simulated by DiffPD [151], a differentiable FEM simulator. Figure 6.2 visualizes the conversion between the implicit and explicit representation. Triangle surface mesh and tetrahedral volume meshes could be explored in future work.

Physics Engine. From the input video of a dynamic scene, we can extract a mesh sequence $\{\mathbf{m}_i\}_{i=1}^N$ with consecutive motion. An individual mesh model $\mathbf{m}_i$ is described by its vertices and hexahedral elements $\{\mathbf{Q}_i \in \mathbb{R}^{n_v \times 3}, \mathbf{E}_i \in \mathbb{N}^{n_e \times 8}\}$, where the rows of $\mathbf{Q}_i$ are the 3D coordinates of n_v vertices, and each row in $\mathbf{E}_i$ records the indices of eight vertices that form a hexahedron. We focus on modeling dynamic objects as soft bodies in this work. Compared to rigid bodies, soft bodies offer more expressiveness to model a wide range of deformable objects that are important in applications like physically-based mixed reality and digital twins. As our physics engine, we use Projective Dynamics [74] implemented by DiffPD [151]. The governing equation can be written as

$$\mathbf{M}(\mathbf{Q}_{i+1} - \mathbf{Q}_i - h\dot{\mathbf{Q}}_i) = h^2(\nabla E(\mathbf{Q}_{i+1}, \theta_{physics}) + \mathbf{f}_{ext}), \quad (6.3.2.2)$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{Q}_i$ is the vertex locations at frame i , h is the time step, $\dot{\mathbf{Q}}_i$ is the velocity, E is the potential energy due to deformation, $\theta_{physics}$ are the physics parameters, and $\mathbf{f}_{ext}$ are the external forces. Note that the gradient of the potential energy ∇E can be interpreted as the internal forces within the soft materials. This system uses implicit time integration for

increased numerical stability, thus ∇E is defined on the frame $i + 1$. Objects are modeled by homogeneous co-rotated materials, where the material parameters $\theta_{physics}$ include Young's modulus and Poisson's ratio. A detailed formulation of the elastic energy E can be found in the supplementary materials of [151].

System identification can be valuable for constructing faithful digital twins, producing plausible animations, and control of dynamical systems, but estimating states $\mathbf{Q}$ and $\dot{\mathbf{Q}}$, and parameters $\theta_{physics}$ and $\mathbf{f}_{ext}$ from only a monocular video is challenging due to the high-dimensionality and nonlinearity of the system. Recent progress in differentiable physics tackles this challenge by making the entire simulation differentiable. Gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{Q}_i}$, $\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{Q}}_i}$, $\frac{\partial \mathcal{L}}{\partial \theta_{physics}}$, and $\frac{\partial \mathcal{L}}{\partial \mathbf{f}_{ext}}$, where $\mathcal{L}$ is a user-defined scalar loss function, are all available, and allow us to estimate the parameters using a gradient-based optimizer like Adam [217].

Parameter Estimation. Running one simulation

step forward in time from the i -th frame gives us an updated model at frame $i + 1$, dependent

on parameters $\theta_{physics}$: $\mathbf{m}'_{i+1}(\theta_{physics}) =$

$sim(\theta_{physics}, \mathbf{m}_i, i \rightarrow i + 1)$. By simulating

the reconstructed meshes in the differentiable

physics engine $sim(\theta_{physics}, \cdot)$ and comparing

with the ground truth reconstruction results as

a supervision signal, we optimize the physics

parameters $\theta_{physics}$ using gradient-based methods.

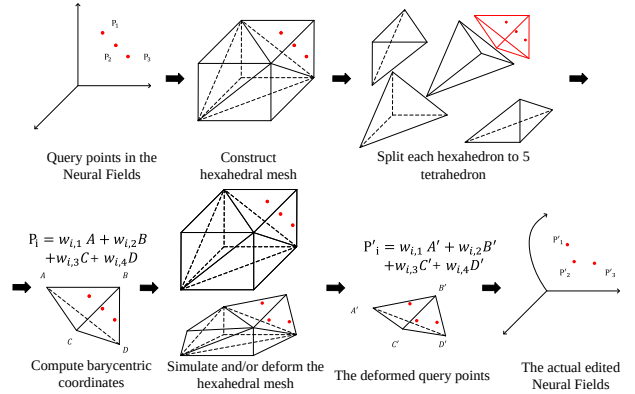


Figure 6.2: **The editing pipeline.** Users can edit the neural fields by operating on a hexahedral mesh.

One idea for a suitable loss function is to use a metric like Chamfer Distance $distance(\mathbf{m}_{i+1}, \mathbf{m}'_{i+1}(\theta_{physics}))$

between the extracted and simulated mesh $\mathbf{m}_{i+1}$ and $\mathbf{m}'_{i+1}(\theta_{physics})$, searching for the $\theta_{physics}$ that minimizes the distance. However, this strategy requires sampling and mesh reconstruction for every time step, which is computationally expensive. An ablation study and profiling numbers can be found in Figure 6.10. Additionally, the reconstructed mesh might vary significantly across time steps, thus cannot guarantee point-wise correspondence, making the dynamics discontinuous and non-differentiable w.r.t. the networks.

To save sampling time and make the gradient well-defined, we design a cycle-consistency loss to compute the distance between the simulation results and neural fields. Starting from an arbitrary mesh $\mathbf{m}_i$ sampled at the i -th frame, we run the differentiable simulation for j steps and get $\mathbf{m}'_{i+j}(\theta_{physics}) = sim(\theta_{physics}, \mathbf{m}_i, i \rightarrow i + j)$. When $\mathbf{m}'_{i+j}(\theta_{physics})$ is deformed back to the reference frame using the learned motion field, the result should correspond to the original shape $\mathbf{m}_i$. Since $\mathbf{m}'_{i+j}(\theta_{physics})$ and $\mathbf{m}_i$ share a common topology, an exact point-wise correspondence exists, and we can directly subtract their point clouds to obtain their distance:

$$\mathcal{L}_{physics} = \left\| (\mathbf{m}_i + \mathbf{b}_{\theta_{motion}}^{\theta_{rigidity}}(\mathbf{l}_i, \mathbf{m}_i)) - (\mathbf{m}'_{i+j}(\theta_{physics}) + \mathbf{b}_{\theta_{motion}}^{\theta_{rigidity}}(\mathbf{l}_{i+j}, \mathbf{m}'_{i+j}(\theta_{physics}))) \right\|_2 \quad (6.3.2.3)$$

where $\mathbf{b}(\mathbf{l}_i, \mathbf{m}_i)$ is the motion field at frame i that maps mesh $\mathbf{m}_i$ back to the reference frame.

$\mathcal{L}_{physics}$ is differentiable w.r.t. the motion network, rigidity network, and physics parameters.

Importantly, independent of the simulation duration, we only need to sample $\mathbf{m}_i$ once, and it can be used as a reference point to compare all other frames.

Scene Editing. The simulator in our pipeline imposes a strong physics prior for forecasting motion and modifying dynamics. Figure 6.2 shows how we edit the signed distance and color fields through the hexahedral mesh. Based on hexahedral mesh $\mathbf{m}_i$ extracted from the i -th frame, we can simulate it to get a modified state $\mathbf{m}'_{i+1}$. In order to render the modified scene, we add an additional ‘bending’ layer $\mathbf{b}_{edit}(\mathbf{m}'_i, \cdot)$ on top of the primary motion field $\mathbf{b}(\mathbf{l}_i, \cdot)$.

When querying a ray point $\mathbf{p}'$ to render the modified scene, we first check if the point is displaced by our editing operation, i.e. if it is located within the hexahedral mesh $\mathbf{m}'_i$. The inside-outside-test would be trivial if all hexahedra in $\mathbf{m}'_i$ remained in a regular grid, however, they may have deformed during simulation. Thus, we split each hexahedron into five tetrahedra – a query point $\mathbf{p}'$ is inside the hexahedron if and only if it is inside one of the tetrahedra. Following [218], we decide whether $\mathbf{p}'$ is inside a tetrahedron by its barycentric coordinates. For a tetrahedron consisting of four vertices $\{\mathbf{p}'_i = (x'_i, y'_i, z'_i)^\top\}_{i=1}^4$, the barycentric coordinates $\{c_i\}_{i=1}^4$ of $\mathbf{p}' = (x', y', z')^\top$ can be computed as $c_i = Det_i / Det_0$, where:

$$Det_0 = \begin{vmatrix} x'_1 & y'_1 & z'_1 & 1 \\ x'_2 & y'_2 & z'_2 & 1 \\ x'_3 & y'_3 & z'_3 & 1 \\ x'_4 & y'_4 & z'_4 & 1 \end{vmatrix}, Det_1 = \begin{vmatrix} x' & y' & z' & 1 \\ x'_2 & y'_2 & z'_2 & 1 \\ x'_3 & y'_3 & z'_3 & 1 \\ x'_4 & y'_4 & z'_4 & 1 \end{vmatrix}, Det_2 = \begin{vmatrix} x'_1 & y'_1 & z'_1 & 1 \\ x' & y' & z' & 1 \\ x'_3 & y'_3 & z'_3 & 1 \\ x'_4 & y'_4 & z'_4 & 1 \end{vmatrix}, \dots, \quad (6.3.2.4)$$

and Det_i is obtained by substituting the i -th row of Det_0 with $(x', y', z', 1)$. $\mathbf{p}'$ is inside or on the surface of this tetrahedron if and only if $0 \leq c_i \leq 1$ holds for $i = 1, 2, 3, 4$. To determine tetrahedra to check for a given point, we use k-nearest neighbors $\mathbf{Q}'_{\mathbf{p}'} = KNN(\mathbf{m}'_i, \mathbf{p}')$ to locate mesh vertices $\mathbf{Q}'_{\mathbf{p}'}$ nearest to query point $\mathbf{p}$, and select tetrahedra that contain those vertices.

we train for physics parameters. The total loss function is:

$$\mathcal{L}_{total} = \mathcal{L}_{color} + w_{Eik}\mathcal{L}_{Eikonal} + w_{off}\mathcal{L}_{offset} + w_{div}\mathcal{L}_{divergence} + w_{phy}\mathcal{L}_{physics} \quad (6.3.3.1)$$

By default, we set $w_{Eik} = 0.1$, $w_{off} = 50.0$, and $w_{div} = 2.0$, though these weights may be further tuned depending on the scene. Figure 6.3 demonstrates the data and flow of gradients during training. In our proposed cycle-consistency physics loss, the rendering and simulation modules are connected by the rigidity and motion networks. Instead of jointly training all the parameters together, we adopt a sequential training strategy where the rendering networks are first trained to optimize $\mathcal{L}_{total}$ for 300,000 epochs, and the physics parameters are then trained to optimize $\mathcal{L}_{physics}$ for 100 epochs. The physics module is trained independently after the geometry networks converge because the differentiable simulation is much more expensive than querying of MLPs. An ablation study for the efficiency of our training strategy is conducted in Section 6.4.5. In future work it would be interesting to try a faster simulation engine which could be incorporated as a physics prior during training of the rendering networks. A full training cycle typically takes around 18 hours on an NVIDIA A5000 GPU. For the training data (used with consent), we note that the background must contain sufficient detail, or the Structure from Motion implementation by COLMAP [220, 221] could fail.

Method	Sitting			Standing			Basketball			Hand		
	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑
D-NeRF	0.163	0.914	34.217	0.251	0.868	33.036	0.332	0.831	32.803	0.165	0.885	34.146
NeuS	0.239	0.911	34.563	0.301	0.829	32.888	0.330	0.862	34.440	0.305	0.908	34.485
NRNeRF	0.223	0.892	32.019	0.269	0.863	31.549	0.340	0.830	32.411	0.205	0.913	33.999
Ours	0.166	0.937	34.563	0.250	0.869	32.881	0.235	0.912	35.974	0.225	0.915	34.710

Table 6.1: **Quantitative evaluation of video reconstruction.** We evaluate our method on both real-world and synthetic videos, outperforming alternative methods in most cases.

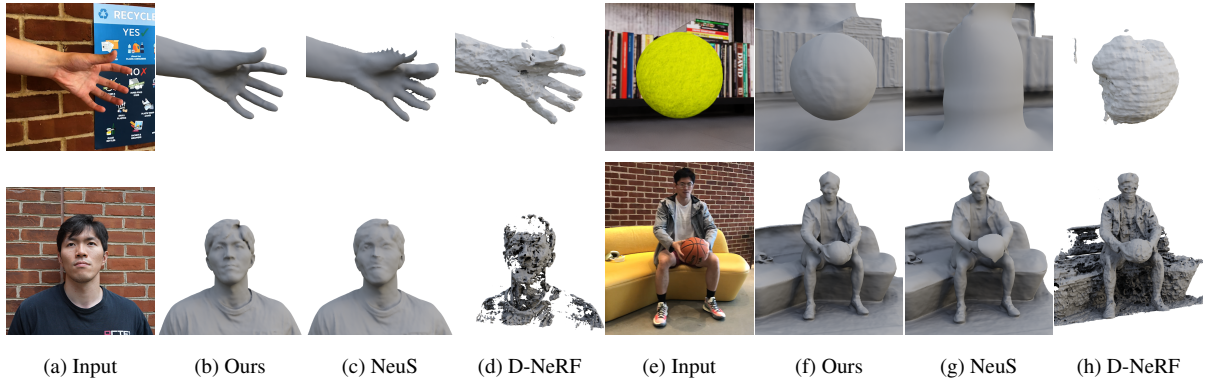


Figure 6.4: **Surface reconstruction from dynamic videos.** Our method achieves higher-quality, more detailed reconstruction (e.g. on muscles, faces, and books) than other approaches.

6.4 Experiments

6.4.1 Dynamic Geometry Reconstruction

We visualize mesh reconstruction quality of dynamic scenes in Figure 6.4. Table 6.1 shows quantitative results. Our method frequently outperforms competitive approaches. Additional geometry reconstruction comparisons can be found in Appendix D.

6.4.2 Novel View Synthesis

We simulate four cases of deformable objects moving in a room. We render each scene with two different camera trajectories, so that one path may be used for training, while the other is used as a ground truth to evaluate novel view synthesis. Quantitative results in Table 6.2 show that our method achieves the best synthesis quality in all metrics. Datasets and results are shown in Figure 6.5. (a) and (b) are frames from the training videos. (c) is taken at the same time step

Method	Tennis Ball			Basketball			Blue Ball			Blue Cube		
	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑
NRNeRF	0.455	0.566	30.286	0.473	0.550	29.684	0.409	0.479	30.152	0.620	0.398	29.044
D-NeRF	0.596	0.392	29.170	0.565	0.425	29.073	0.596	0.317	28.937	0.775	0.229	27.855
Ours	0.336	0.632	31.103	0.368	0.625	30.568	0.334	0.513	30.407	0.589	0.449	29.997

Table 6.2: **Quantitative evaluation of novel view synthesis.** In this experiment, we evaluate a novel camera trajectory different from the training data. Even in this highly under-constrained task, our method benefits from the consistency of the geometry and significantly outperforms S.O.T.A. using all metrics.

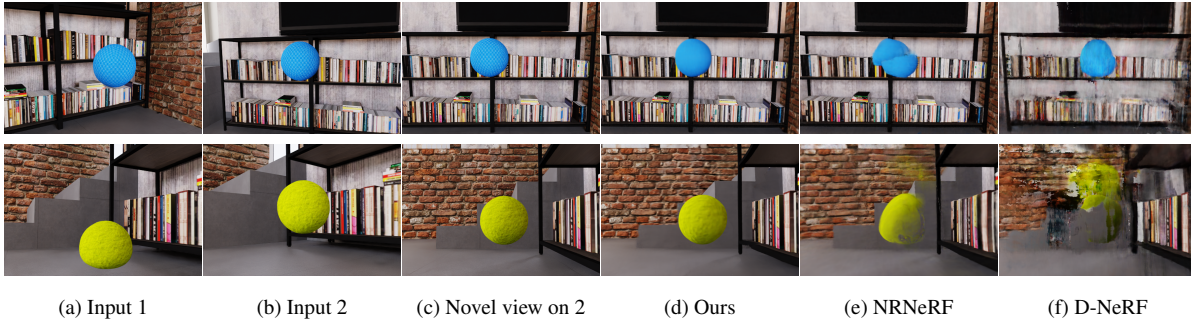


Figure 6.5: **Qualitative evaluation of geometry reconstruction and novel view synthesis.** (a, b) Frames from the training videos. (c) Ground truth evaluation view. Consistent with numerical results in Table 6.2, our method generates images from novel views with higher quality than previous methods.

as (b), but it is a novel view from the test path. Notice that our method successfully synthesizes the dynamic scene from a novel view, while other methods struggle to reconstruct geometry and texture accurately.



Figure 6.6: **Parameter estimation and editing by differentiable physics.** NeuPhysics can estimate the acceleration and Young’s Modulus, generate new videos with different initial velocities and material parameters.

6.4.3 Parameter Estimation

In these experiments, we first use our network to reconstruct the dynamic scene, then apply the embedded differentiable simulator to estimate physical parameters. Finally, we edit the physical parameters to generate novel video frames. More visual results can be found in the supplementary video. Figure 6.6 shows the initial frame of a bouncing ball sequence, where the ball is released from a height and collides with the ground. The ball is modeled by co-rotated material. Using DiffPD [151], we estimate both Young’s modulus and vertical acceleration in two separate optimization tasks. The physics loss is minimized for 100 epochs (as described in Section 6.3.2) using Adam [217] optimizer with a learning rate of 0.01. We initialize Young’s modulus to $2 \times 10^5 Pa$ and acceleration to $0m/s^2$, and by the end of optimization, have estimated Young’s modulus at $2.96 \times 10^6 Pa$ and acceleration at $3.78m/s^2$. Having identified accurate physics parameters, we may choose to modify some of them from their true values, and generate videos that contain modified physical properties from the original on which they are based. For example, the original ball has no horizontal speed, so we add $0.3m/s$ left and right initial velocities in Figure 6.6 (b) and (c), respectively. We also decrease the Young’s modulus in Figure 6.6 (d), (e), and (f) from $2 \times 10^7 Pa$ to $2 \times 10^5 Pa$, where the deformation is getting more significant.

6.4.4 Editing

In this section, we demonstrate how our hybrid representation provides a flexible interface for users to directly modify the geometry on the 3D hexahedral mesh and volume render the results.

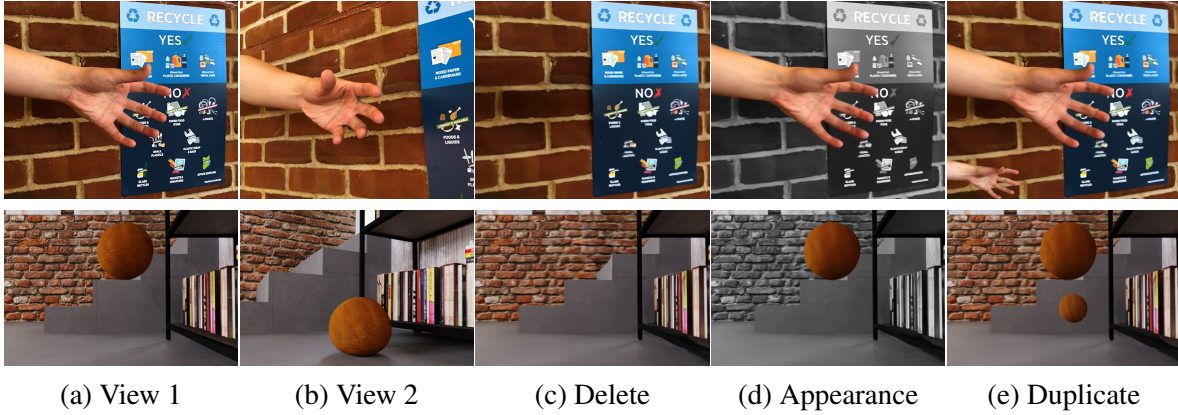


Figure 6.7: **Geometry and appearance editing in videos.** Our method can delete, add, move, or change the appearance. Editing in the 3D space can provide users with realistic and clean results.

Region of Interest. First, we select a region of interest that we wish to edit. Our system naturally provides the scene rigidity mask and SDF values, so a foreground area $\mathcal{A}_i$ at the i -th frame can be computed as Equation 6.3.2.1. If users require a more precisely defined region of interest, a custom bounding box or even more complex hexahedra bounding volume $\mathcal{A}'$ can be provided. A hexahedra mesh $\mathbf{m}$ can be reconstructed from the final region of interest, $\mathcal{A}' \cap \mathcal{A}_i$.

Delete. Figure 6.7 (a) and (b) are two distinct time steps from a given input video. In one, a hand flexes and deforms, and in the second, an elastic ball bounces. First, we delete the foreground from View 1 by simply setting the SDF value to a positive number (e.g. 1.0) for the query points inside the hexahedral mesh $\mathbf{m}$. Since we assume a bell-shaped density distribution centered at 0 [88], an area with large SDF value is less likely to be sampled, and therefore appears transparent. After removing the foreground from View 1, even though some parts of the background were occluded in the original, the other views of the scene provide sufficient textural/structural information to complete the missing background. Since editing takes place entirely in 3D space, the resulting 2D images feel sharp and physically grounded, as shown in (c).

Appearance. We can also modify the output of the color network to change the appearance of a

specific region. For example, we might like to emphasize the moving foreground of a video by desaturating the background. This is easily done by changing the original color $[r, g, b] = \mathbf{c}_{\theta_c}(\mathbf{p})$ to be $[\frac{r+g+b}{3}, \frac{r+g+b}{3}, \frac{r+g+b}{3}]$ for all the query points outside $\mathbf{m}$. Results are shown in Figure 6.7 (d).

Duplicate. Translation and scaling are basic operations in graphic design software. With our pipeline, users are able to copy, move, and scale the hexahedral mesh $\mathbf{m}$, creating a new version $\mathbf{m}'$. During volume rendering, the SDF and color values of the points $\mathbf{p}'$ inside $\mathbf{m}'$ are set equal to their counterparts $\mathbf{p}$ in $\mathbf{m}$. In Figure 6.7 (d), we duplicate an existing object and modify its size and position.

6.4.5 Ablation Study

In this section, we conduct several ablation studies using the bouncing ball example to justify our network design choices and training strategies.

Sequential Vs. Joint Optimization. Our method learns the scene sequentially by first optimizing the rendering networks, followed by the physics parameters. A seemingly more concise strategy might be to jointly train all parameters from scratch. Unfortunately, the joint training strategy is impractical for two reasons: (1) the table in Figure 6.8 indicates that the physics module converges rapidly thanks to the effectiveness of the differentiable simulator, but

	Rendering Module	Physics Module
Epoch	300,000	20
Time per epoch	~ 0.22 s	~ 70 s
Total time	~ 18 h	~ 0.5 h

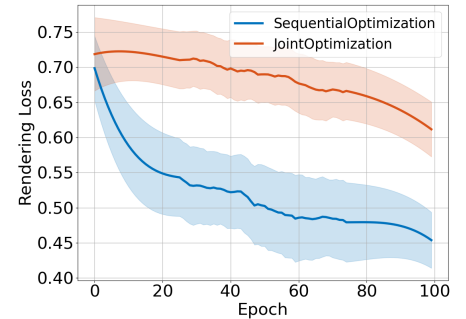


Figure 6.8: Joint training can be time-consuming and converges slowly.

running the forward and backward simulation is costly (~ 70 s) compared to querying the rendering networks (~ 0.22 s) – training the rendering module and physics module together would require many more iterations (300,000 epochs), with each taking much longer (~ 70 sec per epoch), making the training infeasible, and (2) before the rendering module converges and provides satisfactory geometry estimation, the geometry makes little sense for simulation, so gradients flowing from the $\mathcal{L}_{physics}$ do not help the optimization. The smoothed plot in Figure 6.8 indicates that joint training slows down the rendering optimization early on. Sequential optimization is therefore more practical.

We conducted two additional tests to study the relationship between the rendering and physics modules. First, during the sequential optimization, we also update the rendering networks. This harms the rendering quality (increases loss from 0.03 to 0.10) and does not help physics parameter estimation. Second, we attempt to optimize the rendered image from physics simulation to match the ground truth image. We found this process to be inefficient (2 minutes per image) and observed no significant benefits.

Physics Module. Within our sequential optimization strategy, there are still multiple options for physical parameter estimation. In our pipeline, we reconstruct the volume mesh $\mathbf{m}_1$ of the first frame, then predict the following mesh sequence $\mathbf{m}'_i = sim(\theta_{physics}, \mathbf{m}_1, 1 \rightarrow i)$ with

the differentiable simulator. The physics loss for each frame can be computed in a cycle-consistency manner as in Equation 6.3.2.3. Our proposed cycle-consistency loss naturally guarantees a point-wise correspondence for each vertex, and does not require mesh reconstruction for every time step. For each epoch, we only reconstruct the volume mesh for the first frame, which reduces computation and makes the loss differentiable w.r.t. the rendering networks.

A more direct physics loss could rely on reconstructing the geometry of each frame and computing the Chamfer distance between the simulated and reconstructed meshes. We compare the two strategies in Figure 6.10. ‘Ours’ is optimized with our proposed cycle-consistency loss $\mathcal{L}_{physics}$ and ‘Chamfer’ is optimized with the Chamfer distance. (c) shows the value of *acceleration*, the optimization variable. (a) and (b) indicate that the losses have comparable performance in both metrics. However, ‘Chamfer’ takes significantly longer since every frame must be reconstructed. Our proposed cycle-consistency loss runs faster and maintains differentiability between the rendering and physics simulation module. Conceptually, decoupling the NeRF and differentiable physics has another drawback: a mesh reconstructed purely from SDF cannot capture the interior *stress and deformation*. Our proposed cycle-consistency loss addresses this concern, as it takes advantage of the learned motion field, so it is capable of finding the point-wise correspondence and describing the deformation.

Rigidity Map. Finding the correct dynamic foreground object is a prerequisite for physical

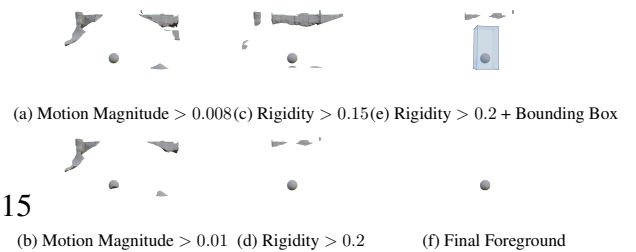


Figure 6.9: **Region of Interest.** Motion magnitude (a b) is an intuitive criterion for

simulation. The raw output of the SDF field can only reconstruct a global scene, so the rigidity map is used to segment the dynamic part of the scene (assumed to be the foreground in this case). The rigidity map collects the motion information from all frames and assigns a high rigidity mask value to dynamic regions. Figure 6.9 (c, d) visualizes the rough segmentation results from using different rigidity thresholds. We empirically find 0.2 to be a reasonable default threshold for separating the foreground from background. If the rigidity map does not provide clean separation, users may manually specify a bounding box covering the moving object swept volume, and get the final foreground as shown in (e, f). An alternative segmentation approach is to use the magnitude of the motion field as a criterion to separate dynamic foreground, i.e. large motion area corresponds to dynamic objects (and the converse). However, in some frames, dynamic parts might only contain small offsets from the canonical frame. For example, a cyclic bouncing ball could overlap with the canonical position and thus have offset values of zero, even though it should be classified as a dynamic area. Figure 6.9 (a) and (b) show results of filtering the scene using motion magnitude. It is not a good strategy, since a large portion of the background remains even as the ball is partially clipped off in (b).

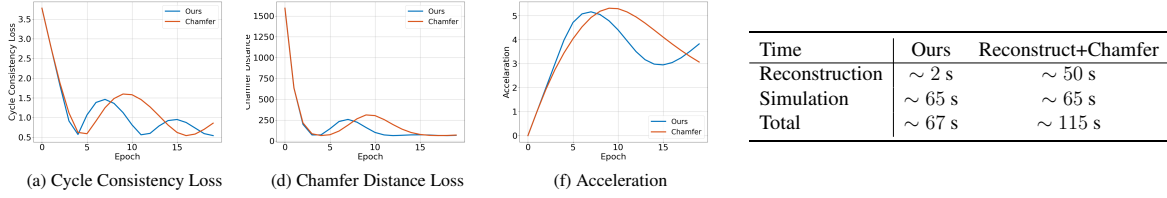


Figure 6.10: **Compare with optimizing Chamfer Distance.** An alternative to our proposed cycle-consistency loss is to use Chamfer Distance for the physics module. The two metrics have similar effects but our method can save much reconstruction time.

6.5 Conclusion

In this chapter, we design an end-to-end pipeline that can successfully estimate and enable physically-based editing of geometry, appearance, and physics of a dynamic scene from a single video. Some possible future directions include incorporating semantic priors to further constrain the optimization, and leveraging pre-trained networks [222] as supervision to further regularize training. For example, optical flow / scene flow networks could be used as a motion detection signal, while a depth estimation network could help to initialize the signed distance field. Another useful future goal would be to estimate texture [223] and UV maps directly, allowing artists to construct 3D models in standard formats with less effort. Training and inference could also be accelerated dramatically by adopting a multi-resolution hash encoding [224]. Finally, we could couple this differentiable framework with other types of dynamics, to support a wider range of objects and scenes, e.g. cloth, tetrahedra mesh [212], or even robots [214].

Acknowledgements. This research is supported in part by Dr. Barry Mersky and Capital One E-Nnovate Endowed Professorships, and ARL Cooperative Agreement W911NF2120076.

Chapter 7: Dynamic Mesh-Aware Radiance Fields

7.1 Introduction

Creating high-quality 3D environments suitable for photorealistic rendering entails labor-intensive manual work carried out by skilled 3D artists. Neural Radiance Fields (NeRF) [79] provide a convenient way to capture a volumetric representation of a complex, real-world scene, paving the way for high-quality novel view synthesis and interactive photorealistic rendering [224]. These qualities make NeRF exceptionally adept at modeling background environments. On the other hand, existing methods for physically-based simulation and rendering of complex material and lighting effects are primarily based on *surface mesh* representations. Integrating neural field representations with well-established traditional graphics pipelines opens up many possibilities in VR/AR, interactive gaming, virtual tourism, education, training, and computer animation.

Volume rendering [225] has demonstrated its capability to produce visually captivating results for participating media [226]. However, integrating NN-based NeRF into this pipeline while maintaining realistic lighting effects such as shadows, reflections, refractions, and more, remains a relatively unexplored area. In terms of simulation, while the geometry of NeRF is implicit in its density field, it lacks a well-defined surface representation, making it difficult to detect and resolve collisions. Recent works have delved into enhancing the integration between NeRF and meshes, aiming to combine the photorealistic capabilities of NeRF with the versatility

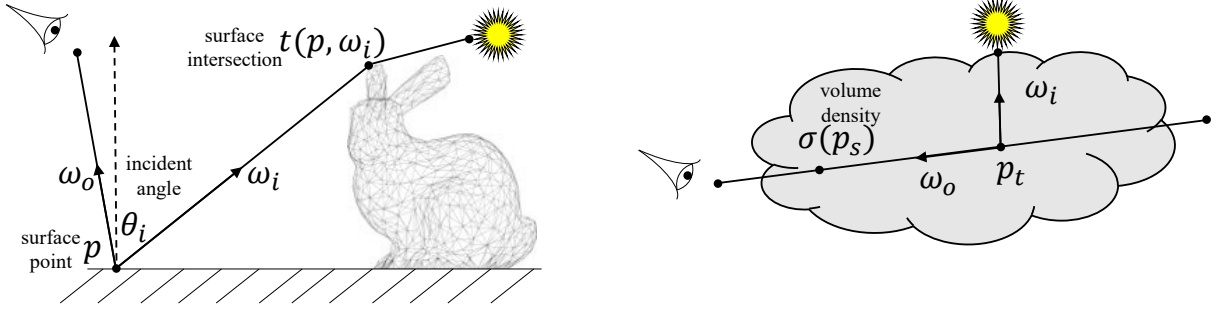


Figure 7.1: **Light transport** on the surface (*left*) and in the medium (*right*).

of meshes for rendering and simulation. Neural implicit surfaces [88, 89, 90, 227] are represented as learned Signed Distance Fields (SDF) within the NeRF framework. Meanwhile, methods like IRON [84] and NVDiffRec [228] extract explicit, textured meshes that are directly compatible with path tracing, offering practical benefits at the expense of a lossy discretization. Nerfstudio [229] renders NeRF and meshes separately, then composites the render passes with an occlusion mask. Unfortunately, this decoupled rendering approach offers no way to exploit the lighting and appearance information encoded in the NeRF volume to affect the rendered mesh appearance. Figure 7.5 visually compares our hybrid method to naively combining NeRF and surface rendering, and pure surface rendering.

We introduce a hybrid graphics pipeline that integrates the rendering and simulation of neural fields and meshes. for both representations, we consider lighting effects and contact handling for physical interaction. By unifying NeRF volume rendering and path tracing within the linear RGB space, we discover their Light Transport Equations exhibit similarities in terms of variables, forms, and principles. Leveraging their shared light transport behavior, we devise update rules for radiance and throughput variables, enabling seamless integration between NeRF and meshes. To incorporate shadows onto the NeRF, we employ differentiable surface rendering techniques [230] to estimate light sources and introduce secondary shadow rays during the ray

marching process to determine visibility. Consequently, the NeRF rendering equation is modified to include a point-wise shadow mask.

For simulation, we adopt SDFs to represent geometry of neural fields, which is advantageous for physical contact handling and collision resolution. We then use position-based dynamics [231] for time integration. Our efficient hybrid rendering and simulation system is implemented in CUDA. To enhance usability, we have also incorporated user-friendly Python interfaces. In summary, the key contributions of this work are:

- A two-way coupling between NeRF and surface representations for rendering and simulation.
- Integration with HDR data which can unify the color space of the path tracer and NeRF, with a strategy to estimate light sources and cast shadows on NeRF.
- An efficient rendering procedure that alternates ray marching and path tracing steps by blending the Light Transport Equations for both NeRF and meshes.
- An interactive, easy-to-use implementation with a high-level Python interface that connects the low-level rendering and simulation GPU kernels.

7.2 Related Work

7.2.1 Neural Fields and Surface Representations

Rendering of participating media has been extensively studied [226, 232] in classic graphics pipelines [233, 234, 235, 236]. In recent years, significant advancements have been made in this area [237, 238], yielding remarkable visual outcomes. Our work aims to expand upon this progress by incorporating the emergent Neural Radiance Fields (NeRF) [79], which have gained

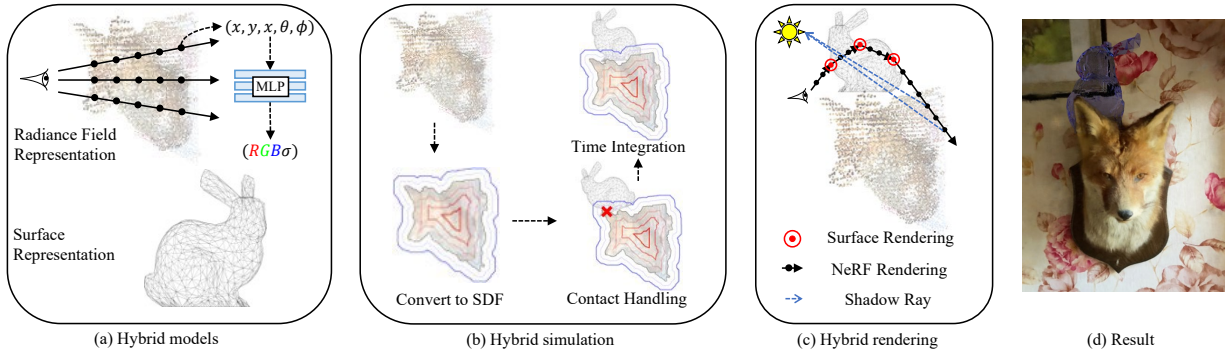


Figure 7.2: Pipeline overview. Our method takes an optimized radiance field model and surface meshes as inputs. We can run a physics simulation between the NeRF and meshes. The updated mesh vertices and NeRF transformations are synchronized to the renderer, which uses Monte Carlo simulation to sample ray paths. As the ray travels through space, it alternates between surface rendering (ray-tracing) and NeRF rendering (ray-marching), both updating its radiance.

substantial popularity, into this exciting domain. Within the volume rendering framework, NeRF bakes the plenoptic function and volumetric density into spatial points. These points can be effectively parameterized by an an MLP [79], convolutional networks [239], hash grid [224], point cloud [240], voxel [241, 242], or tensors [243]. It allows users to reconstruct photorealistic 3D static [244, 245, 246, 247, 248] or dynamic scenes [101, 104, 109, 110, 249] by casually capturing a few images [250]. Original NeRF takes seconds to render one single frame, while follow-up works have accelerated rendering speed [251, 252, 253, 254, 255, 256, 257, 258, 259, 260].

NeRF simplifies the creation of 3D content compared to classical mesh-based pipelines. However, addressing challenges of editing [103, 261] and decomposing the baked information [262, 263, 264, 265, 266] is not trivial. Effort has been directed toward reconciling the advantages of both NeRF and surface-based paradigms. In rendering, [84] and [228] propose to use surface-based differentiable rendering to reconstruct textured meshes [267] from neural fields. Their reconstructed meshes can be imported to a surface rendering pipeline like Blender [268], but

the original NeRF representation cannot be directly rendered with meshes. For simulation, NeRFEediting [218] proposes to use explicit mesh extracted by [88] to control the deformation of Neural Fields. Qiao et al. [269] further add full dynamics over the extracted tetrahedra mesh. Chu et al. [270] integrates the dynamics of smoke with neural fields. [271] also connects differentiable simulation to NeRF, where the density field and its gradient are used to compute the contact. These methods aim to construct an end-to-end differentiable simulation and rendering pipeline, yet they have yet to couple the rendering.

7.2.2 Scene Editing and Synthesis

Our method enables inserting mesh assets into NeRF models of real-world captures. Editing of existing scenes is an active topic of study. For neural field representations, ray bending [105, 106, 272] is widely used to modify an optimized NeRF. It is possible to delete, add, duplicate, or actuate [273, 274, 275, 276] an area by bending the path of the rendering ray. [277] propose to train a NeRF for each object and compose them into a scene. ClimateNeRF [278] can change weather effects by modifying the density and radiance functions during ray marching. These methods study editing of isolated NeRF models. There are also inverse rendering works that decompose [82, 279] the information baked into NeRF, which can then be used to edit lighting and materials [280]. Such decomposition is useful, but assumes information like a priori knowledge of light sources, or synthetic scenes. They do not address inserting mesh into NeRF scenes. Besides NeRF, [281] inserts a virtual object into existing images by estimating the geometry and light source in the existing image. [282] insert vehicles in street scenes by warping textured cars using predicted 3D poses.

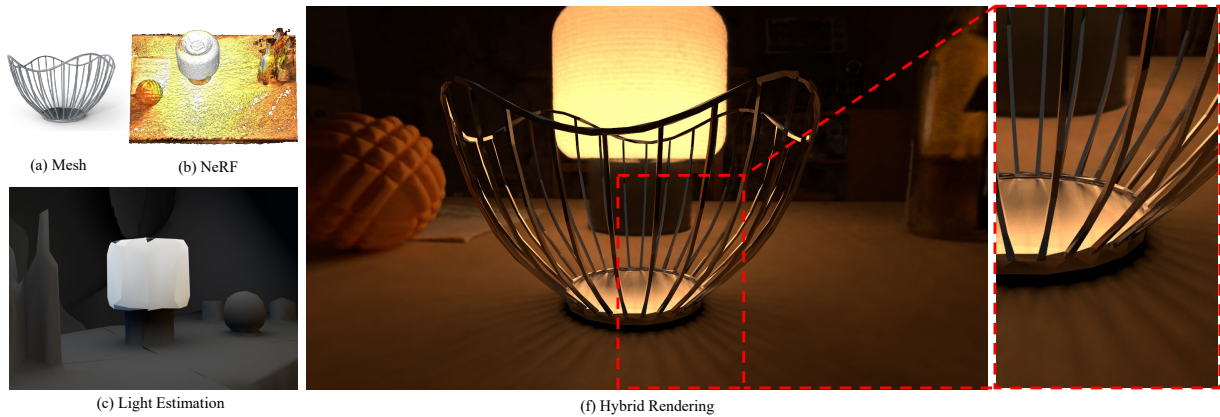


Figure 7.3: **Shadow Casting.** We estimate the geometry and light source of the scene and insert a metal basket onto the desk. Our pipeline can render the realistic reflection and shadow effects caused by the synthetic mesh.

7.3 Method

In this section, we describe how radiance fields and polygonal meshes can be integrated into a unified rendering and simulation pipeline, an overview of which is visualized in Figure 7.2.

7.3.1 Rendering

NeRF can photorealistically reconstruct a 3D scene from a set of images, making it an appealing candidate for modeling environments and potentially valuable for traditional surface-based rendering. One possible approach to bridge these disparate representations is to render the meshes and NeRF volume in separate passes, and composite the results together in 2D image space. However, compositing in image space is susceptible to incorrect occlusion masks and inaccurate lighting. A more physically principled approach to this problem is identifying and exploiting the similarities in their respective light transport equations, which directly allows the radiance field and mesh to be incorporated in 3D space.

Surface Rendering Equation. The Light Transport Equation (LTE) for surface rendering is:

$$L(p, \omega_o) = L_e(p, \omega_o) + L_r(p, \omega_o) \quad (7.3.1.1)$$

$$L_r(p, \omega_o) = \int_{S^2} f_s(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| d\omega_i \quad (7.3.1.2)$$

$$L_i(p, \omega_i) = L(t(p, \omega_i), -\omega_i) \quad (7.3.1.3)$$

where p is a surface point; ω_i, ω_o are the directions of incident (incoming) and exitant (outgoing) radiance; S^2 is the unit sphere sampling space for directions; L, L_e, L_i, L_r are the exitant, emitted, incident, and reflected radiance, respectively; θ_i is the angle of incidence of illumination; $f_s(p, \omega_o, \omega_i)$ is the bidirectional scattering distribution function (BSDF); and $t(p, \omega)$ is the ray-casting function that computes the first surface intersected by the ray cast from p in the direction ω .

If a scene is represented solely by surfaces, the LTE in Equation 7.3.1.1 can be solved by Monte Carlo path tracing: for each pixel, a ray is randomly cast from the camera, its path constructed incrementally each time it hits, and bounces off of a surface. A natural but memory-inefficient way to implement this algorithm is to recursively compute Equation 7.3.1.1 and spawn a new ray upon each ray-surface intersection. Noticing that $L_i(p, \omega_i)$ is independent of previous paths, the recursive process can be transformed into a weighted sum of radiance on each ray-surface intersection p_k . These weights $T(p_k)$ are called *throughput*, and they depend on their predecessors' BSDF $f_s(p, \omega_o, \omega_i)$, illumination angle $|\cos \theta_i|$, and probability density function P

of the scattering:

$$T(p_k) = \frac{T(p_{k-1}) \cdot f_s(p_k, \omega_{k-1}, \omega_k) |\cos(\theta_k)|}{P} \quad (7.3.1.4)$$

$T(p_k)$ and $L_i(p_k, \omega)$ are the only variables essential to track and integrate for each bounce.

Volumetric Rendering Equation. The light transport equation for the volumetric medium is:

$$L(p, \omega_o) = \int_{t=0}^{t_f} \exp\left(-\int_{s=0}^t \sigma_t(p_s) ds\right) L_i(p_t, \omega_o) dt \quad (7.3.1.5)$$

$$L_i(p_t, \omega_o) = L_e(p_t, \omega_o) + L_s(p_t, \omega_o) \quad (7.3.1.6)$$

$$L_s(p_t, \omega_o) = \sigma_s(p_t) \int_{S^2} f_p(p_t, \omega_o, \omega_i) L(p_t, \omega_i) d\omega_i \quad (7.3.1.7)$$

where L_s is the (weighted) scattered radiance, σ_t and σ_s are the attenuations and scattering coefficients, f_p is the phase function of the volume, and p_s is on the ray $p_s = p + s \cdot \omega_o$ (similar to p_t). All other terms share the same definition as in surface rendering.

The integral in the volumetric LTE could again be solved using Monte Carlo methods. However, stochastic simulation of volumetric data is more challenging and expensive than surface data. A photon may change direction in a continuous medium, unlike the discrete bounces that occur only at surfaces. Therefore, rather than simulating the path of photons using Monte Carlo sampling, methods like NeRF [79] instead bake the attenuation coefficient $\sigma(p) = \sigma_t(p)$ and view-dependent radiance $r(p, \omega)$ onto each spatial point, and so there is no scattering. This circumvents solving Equation 7.3.1.6, thereby avoiding considering light transport, light sources,

and material properties. Volume rendering under the NeRF formulation becomes:

$$r(p, \omega) = \int_0^{t_f} T(t) \sigma(p_t) r(p_t, \omega) dt, \quad (7.3.1.8)$$

$$T(t) = \exp \left(- \int_0^t \sigma(p_s) ds \right) \quad (7.3.1.9)$$

In Equation 7.3.1.8, the radiance and throughput are being updated, similar to surface rendering. However, note that the volumetric LTE denotes incident radiance at point p from direction ω as $L_i(p, \omega)$, while NeRF denotes the same as $r(p, \omega)$. This terminology is indeed overloaded, as $r(p, \omega)$ in the NeRF formula represents sRGB color, i.e. the result of applying a nonlinear tone-mapping function to the raw radiance value. The terms are related in that $r(p, \omega) = \psi(L_i(p, \omega))$, where $\psi(\cdot)$ represents a tone-mapping function from linear to sRGB color space.

Unifying color space of path tracing and ray marching. The standard NeRF model accumulates sRGB color along rays cast from the camera into the volume: each point’s color is represented by three 8-bit values, one for each color channel. Integrating these colors along the ray (weighted by transmittance) produces a final 8-bit color value, the rendered pixel color, which is compared to the corresponding ground truth 8-bit pixel color to supervise NeRF training.

In contrast, path tracing assumes radiance values are expressed in linear color space. To relate NeRF and surface rendering in a physically meaningful way, they should ideally operate in a standard color space. To reconcile this difference, we train an HDR variant of NeRF, supervised with 32-bit HDR images directly rather than the standard 8-bit NeRF. The resulting HDR NeRF produces a 3-channel radiance in 32-bit *linear color space* at each sampled point. For details regarding HDR data acquisition, preprocessing, and HDR NeRF implementation, see Appendix E. As our focus here is to articulate the advantage of HDR NeRF in the context

of the overall system, we forego a more general discussion of training NeRF in HDR and refer interested readers to [283] for a deep dive. The HDR NeRF rendering equation can thus be written as: $L_i(p, \omega) = \int_0^{t_f} T(t) \sigma(p_t) L_i(p_t, \omega) dt$ where the transmittance term $T(t)$ remains unchanged.

With this simple adjustment, the NeRF equation can now directly relate to the surface rendering equation. If capturing HDR training data is impractical, one can still use standard (LDR) NeRF. Since the NeRF volume acts as the only light source, and the total energy is dissipative during the light transport, $L_i(p, \omega)$ will never exceed the NeRF volume’s maximal radiance. $\psi(\cdot)$ would then degenerate to an identity mapping such that $L(p, \omega) = r(p, \omega)$. In other words, in many practical cases, reasonable visual results could still be obtained if standard NeRF is used with our system, despite the resulting inaccuracy in the light transport simulation.

Estimating light sources with differentiable surface rendering. Sampling light sources for computing the shadow pass requires an approximate representation of the light sources (emitters) of the scene. Note that NeRF’s volume rendering formulation bakes appearance into each point in the volume rather than simulating physically based light transport. To recover an explicit representation of light sources, we first reconstruct the scene’s geometry as a neural SDF using *MonoSDF* [284], from which we extract an explicit mesh. Then, we employ a differentiable path tracer, *Mitsuba3* [230, 285], to estimate a UV Emission Texture for the mesh. We follow the general approach of [286], though we customize the optimization procedure since our goal is to estimate only the light sources, as opposed to a full BRDF estimation (more details about the optimization can be found in Appendix E).

Once the light source estimation has converged, we prune faces whose emission falls below a threshold from the explicit mesh, which is necessary for efficiency, as most faces in the explicit

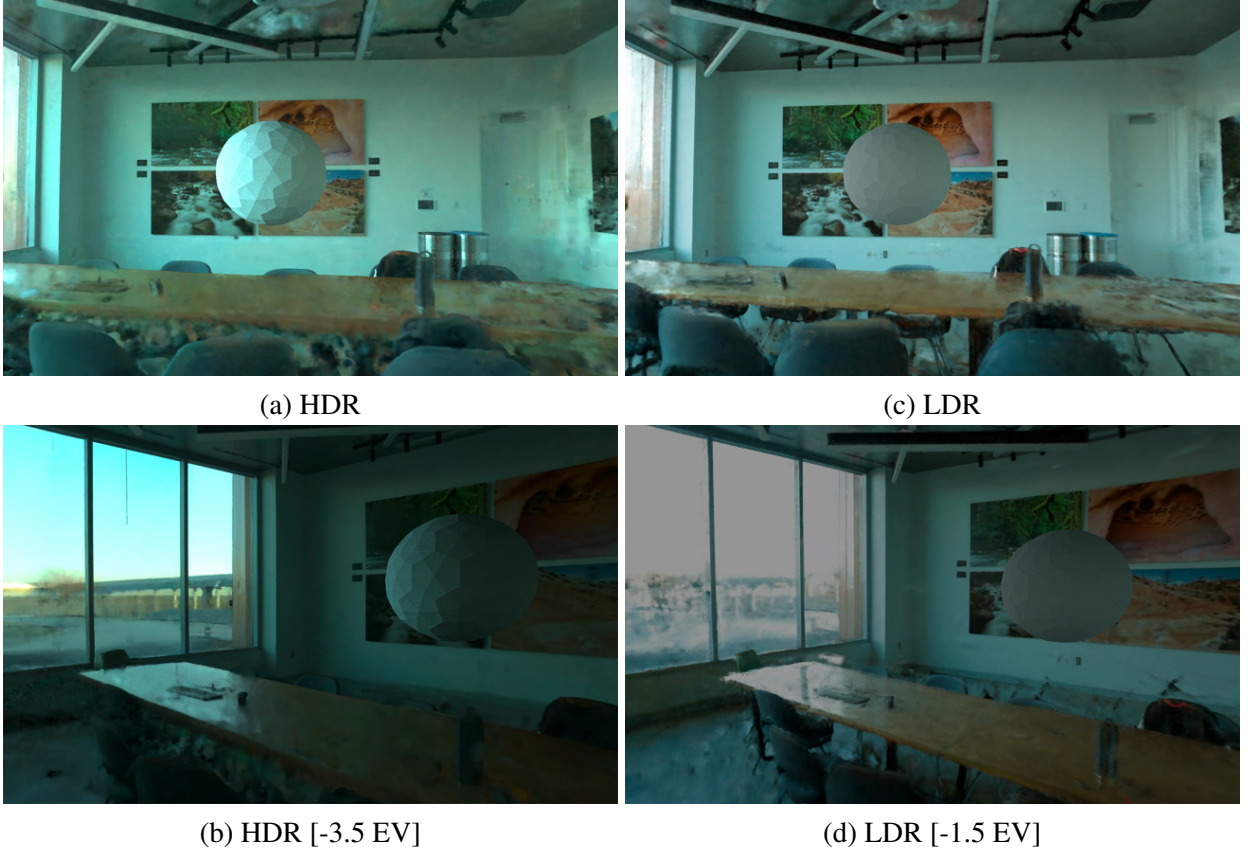


Figure 7.4: **HDR Volumetric Radiance Map.** The diffuse sphere is rendered using our hybrid algorithm. Images rendered using the NeRF trained in 32-bit HDR (a, b) achieve a higher level of lighting realism than those rendered with 8-bit LDR (c, d).

mesh do not emit light. The hybrid renderer then consumes the pruned mesh.

Shadow rays. We query additional rays during ray-marching to cast shadows on NeRF. For each sampled point p_t in NeRF, we shoot a secondary ray from p_t to the light source (see the following subsection for details on estimating lighting sources). If an inserted mesh blocks this ray, then this p_t has a shadow mask $m(p_t) = 1 - r_{src}$, where r_{src} is the intensity of the light source. Non-blocked pixels have $m_{shadow} = 1$. The contribution of this point in Eq. 7.3.1.8 is then $T(t)\sigma(p_t)m(p_t)r(p_t, \omega)$. Fig. 7.3 shows that an inserted metal basket casts shadows on the NeRF desk.

Hybrid Rendering Algorithm Based on the Light Transport Equations mentioned above, we

note that both surface and NeRF rendering integrate the throughput $T(p)$ and radiance $r(p, \omega) = L_i(p, \omega)$. The differences are: (1) Surface rendering updates those values on discrete boundaries while NeRF accumulates them in the continuous space; (2) $T(p)$ and $r(p, \omega)$ are governed by the BSDF parameters in surface rendering, while by neural fields in NeRF. Therefore, we can alternate between the surface and NeRF rendering rules as they travel in space. Algorithm 3 is a summary of the hybrid rendering of NeRF and surface representations:

1. We use Monte Carlo path tracing to sample the ray-surface-intersections $p_0 \rightarrow p_1 \rightarrow \dots \rightarrow p_n$, where p_0 is the camera center, and p_n is the termination of the path. At the beginning of the path, initialize accumulated throughput $T(p_0) = 1$ and radiance $r(p_0, \omega_0) = (0, 0, 0)$. The termination conditions will be discussed in (4).
2. If shadows are needed, we estimate light source geometry and intensity r_{src} with differentiable surface rendering.
3. For each ray segment $p_j \rightarrow p_{j+1}$, we use the ray-marching algorithm to sample and integrate the NeRF medium. For the sampled points p_t on the ray, we shoot a ray from p_t to the light source (if any). If meshes block this point, set its *shadow mask* to be $m(p_t) = 1 - r_{src}$ (or simply a constant close to 0), otherwise $m(p_t) = 1$. Then the throughput and *shadow masked* radiance between surface intersections p_j and p_{j+1} can be computed as,

$$T'(p_{j+1}) = T(p_j) \cdot \exp \left(- \int_{p_t} \sigma(p_t) dt \right) \quad (7.3.1.10)$$

$$L(p_{j+1}, \omega_j) = L(p_j, \omega_j) + \int_{p_t} T(t) \sigma(p_t) m(p_t) r(p_t, \omega) dt \quad (7.3.1.11)$$

where $p_t \in (p_j, p_{j+1}]$ and

$$T(t) = T(p_j) \cdot \exp \left(- \int_0^t \sigma(p_s) ds \right)$$

is also accumulated from p_j .

4. At the end of a ray segment, we reach the interface p_{j+1} where the surface-rendering procedures occur. The direction ω_{j+1} of the next ray is determined by sampling the BSDF, and the weighted illumination and emitted light at this point are added to the radiance:

$$L(p_{j+1}, \omega_{j+1}) = L(p_{j+1}, \omega_j) + T(p_j) L_e(p_{j+1}, \omega_j) \quad (7.3.1.12)$$

The throughput weight is updated as:

$$T(p_{j+1}) = \frac{T'(p_{j+1}) \cdot f_s(p_{j+1}, \omega_j, \omega_{j+1}) |\cos(\theta_{j+1})|}{P} \quad (7.3.1.13)$$

where P is the scattering probability density function.

5. In the end, the ray terminates at (p_e, ω_e) if (1) it runs out of the scene; (2) current throughput $T(p_e)$ is lower than a threshold; or (3) it meets the bounce limit.
6. As the rendering procedure is carried out over a linear 32-bit color space after the path tracing terminates for a given pixel, we can apply a nonlinear tone-mapping function, which we denote as ψ , to map from linear radiance to final sRGB color $r(p_e, \omega_e)$ which is more suitable for displaying on a monitor:

$$r(p_e, \omega_e) = \psi(L(p_e, \omega_e)) \quad (7.3.1.14)$$

Algorithm 3 Hybrid Rendering Pipeline

Require: Meshes and pretrained NeRF of the scene.

Estimate light sources with differentiable surface rendering.

for each pixel (u, v) in parallel **do**

Initialize p_0, ω_0 based on (u, v) and camera center.

Set throughput $T(p_0) = 1$.

Set radiance $L(p_0, \omega_0) = (0, 0, 0)$.

for $j \in \{1, \dots, n_{bounces}\}$ **do**

Cast ray to find next intersection p_j (Eqn. 7.3.1.3).

March along ray $p_{j-1} \rightarrow p_j$.

Cast shadow rays to light sources.

Integrate $T'(p_j)$ and $L(p_j, \omega_{j-1})$ (Eqn. 7.3.1.10, 7.3.1.11).

Sample BSDF at p_j to get next ray direction ω_j

Update $T(p_j)$ and $L(p_j, \omega_j)$ (Eqn. 7.3.1.13, 7.3.1.12).

break if termination conditions satisfied.

(Path tracing endpoint denoted as (p_e, ω_e))

Apply tone-mapping function $r = \psi(L(p_e, \omega_e))$.

7.3.2 Simulation

We incorporate a dynamics simulator that supports rigid bodies, cloth, and deformable solids. Neural fields and meshes can be connected in the simulation pipeline by Signed Distance Fields (SDF) or reconstructed surface mesh. The SDF of the NeRF can be obtained in several ways during pre-processing. On the one hand, existing methods can directly learn the SDF, like NeuS [88] and VolSDF [89]. On the other hand, it is common to set a threshold value for the density field and extract the surface mesh through Marching Cube [216]. And the SDF can be converted from the mesh. The learned SDF has better quality but takes more time. We also implement an Instant-NGP version of NeuS (called NeuS-NGP) and accelerate the original code by more than 10 times. Users can make the trade-off depending on their needs.

We employ extended position-based dynamics (XPBD) [288, 289] to simulate the objects during runtime. We choose this dynamics model because it is fast and can support various

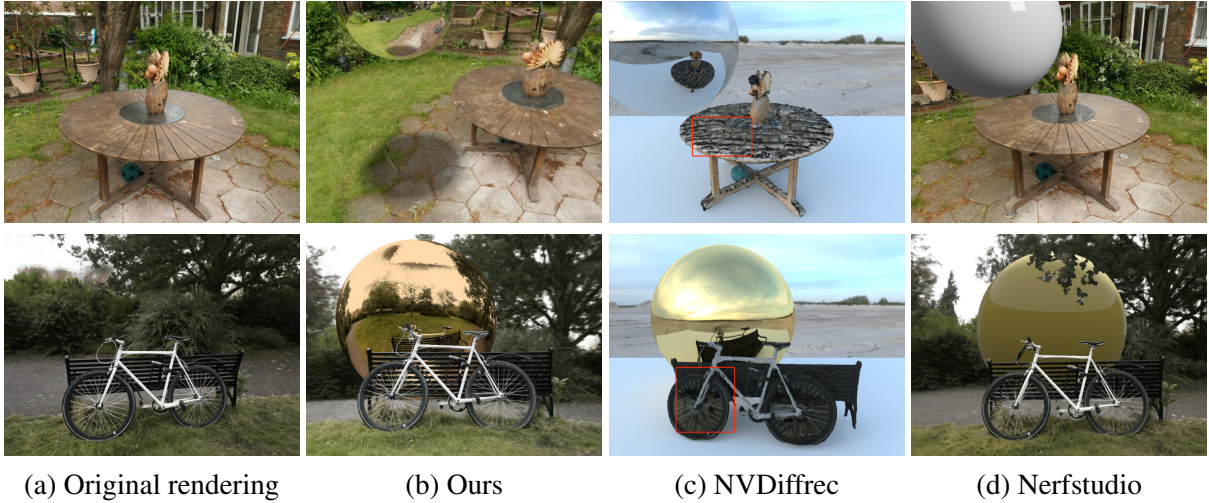


Figure 7.5: **Rendering comparison for virtual object insertion.** We insert a reflective metal ball into the Garden and Bicycle scenes from the Mip-NeRF 360 dataset [287]. (b) Our hybrid method produces results of superior visual quality with fewer artifacts. (c) We extract the foreground mesh using NVDiffRec, then insert the synthetic ball and render using ray tracing. Extracting an explicit mesh object results in noticeable artifacts such as the noisy table surface, and missing thin structures like the bicycle’s wheel and the bench. (d) The 2D compositing workflow of Nerfstudio suffers from non-3D-aware occlusion masks and a limited ability to accommodate realistic interreflection.

physical properties. Collision detection is performed by querying the SDF of all vertices. All of these queries can be computed efficiently in parallel on a GPU. Given a detected collision from the SDF, we can also get the penetration depth and normal, which can be used to compute the contact forces.

In some scenarios, NeRF can represent movable objects (e.g. a scene can be a composition of several NeRF objects [290]) instead of a static background. We can get the homogenous transformation $\mathbf{t} \in \mathcal{R}^{4 \times 4}$ of the NeRF from simulation in each time step, which is used to inverse transform the homogenous coordinates $\mathbf{t}^{-1} \cdot \mathbf{p}$ [105] when querying the color/density and sampling rays in the Instant-NGP hash grid. In Fig. 7.7 (b), we control a ball to interact with a NeRF chair [79] in real-time. The supplementary video further shows how the collision effect changes when the ball and chair have different relative mass and velocities.

7.3.3 Implementation Details

The entire pipeline employs CUDA backends for computation and Python interfaces for interaction. For rendering, NeRF is trained with the default configuration using Instant-NGP [224]. We also implement an instant-NGP [224] version of NeuS [88] for efficient learning of the implicit SDF geometry. The path tracing algorithm is implemented using CUDA, embedded in Instant-NGP’s ray-marching procedures. We incorporate refractive, reflective, and Lambertian BSDF models. Physics simulation utilizes Warp [291], which just-in-time compiles Python code into CUDA kernels. The connection between rendering and simulation is facilitated by a Python interface using pybind11 [164]. Scene parameters can be easily created or modified through config files or Python APIs.

Our method can achieve a runtime of 1 to 40 frames per second, contingent upon the resolution, scene complexity, and dynamics. Figure 7.7 demonstrates a real-time game on a laptop that has been developed within our pipeline. The code for rendering, simulation, and fast SDF learning will be released as open-source software.

7.4 Experiments

7.4.1 Comparisons

Rendering Comparisons. In this section, we compare with other surface-based modeling and rendering methods in the virtual object insertion task. Given a set of images, the traditional graphics pipeline would first reconstruct the surface. However, the 3D reconstruction step will usually introduce tremendous noise and errors. Our technique can directly render the virtual



Figure 7.6: **Qualitative results for simulation.** In addition to rigid body simulation (Fig. 7.7), our method can also simulate soft bodies. (a) is a twisting Neo-Hookean FEM mesh. (b) is a thin shell cloth covering the fox. Please see the supplementary video for more simulation results.

object in the photorealistic 3D scene without meshing the entire scene.

NVdiffrec [228] and IRON [84] are state-of-the-art textured mesh reconstruction methods. They combine neural fields and differentiable rendering methods to estimate the geometry and appearance of the objects from images. However, neither of them works on the full image because the topology of the background is too complex to optimize (e.g. the vegetation). We further provide the per-frame foreground mask to the comparison methods, and NVdiffRec can reconstruct foreground models.

In Figure 7.5 (c), we import the extracted mesh into Blender with a metallic ball and environment map, then render the scene with Cycles, a physically-based render. Both (b) ours and (c) NVDiffRec can model the reflection on the ball, but our object has better surface quality.

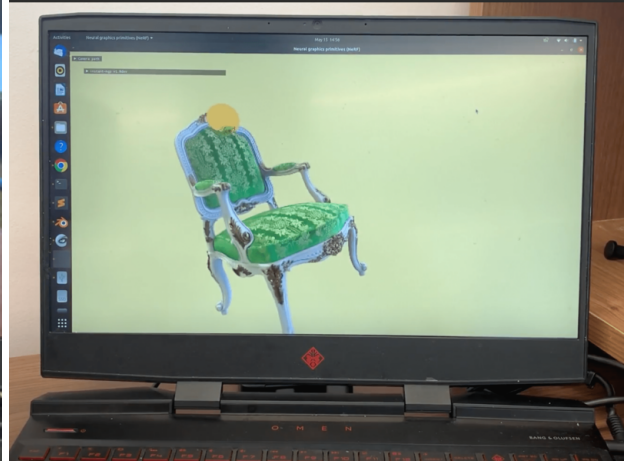
We also compare to Nerfstudio [229], a software library that allows users to train their own NeRF models. The user can export a camera path moving through the trained NeRF scene and a coarsely reconstructed mesh and import these into Blender [268] using the Nerfstudio

Blender plugin. The user may then create their 3D content directly in Blender, using the imported camera path and mesh as a reference, and then render their 3D content, which is then composited in 2D over the rendered NeRF trajectory. However, while the coarse mesh may be used to generate a visibility mask to produce occlusion, this approach is limited in that the coarsely reconstructed mesh is likely not accurate enough to provide clean-looking occlusion results, more complex occlusion situations such as occluding objects at multiple layers of depth will be challenging to generate individual visibility masks, and this requires manual effort to composite the results. Appendix E includes some more quantitative comparisons with those methods. Moreover, LumaAI recently released a closed-source UE plugin, and we run a comparison against it.

Rendering with HDR NeRF. We provide a qualitative comparison between rendering using the standard (LDR) NeRF and its HDR counterpart, shown in Figure 7.4. Notice that in the images rendered with the HDR model, the lighting cast from the environment onto the mesh appears much more faithful to the scene’s true intensity (and therefore directionality). This is also not surprising, considering the longstanding use of High Dynamic Range Image-Based Lighting in the traditional graphics pipeline [292]. HDR images are often created by recovering the unknown nonlinear tone-mapping function from a series of bracketed-exposure LDR images with known exposure duration, then using the inverse function to map the images back to linear color space, and merging them into a single 32-bit result, as introduced by Debevec and Malik [293]. As a result, HDR images are much better suited to capture a scene’s full range of absolute and relative radiance values and avoid highly lossy clipping, which can be especially problematic in very bright parts of an image. HDR NeRF may therefore be interpreted as a volumetric HDR lighting



(a) Static NeRF background



(b) NeRF (chair) as dynamic object

Figure 7.7: Real-time photorealistic gaming on a laptop. Our rendering and simulation engine can be interactive and run in real-time on a laptop with an NVIDIA GeForce RTX 2070 Max-Q GPU. In this example, a user can control the motion of the synthetic glass ball and interact with the background (collision and light effect).

map.

Importantly, our hybrid rendering algorithm allows us to utilize such an HDR volumetric radiance map fully. As an HDR radiance map is particularly useful for representing the indirect bounce lighting of the scene, future work on extending such learned lighting models to direct lighting (e.g. directional or point sources that cast hard shadows) would be a promising direction. Our hybrid rendering algorithm is crucial to enable such an investigation.

Simulation Comparisons. Besides static scenes, our pipeline can also simulate dynamic scenes with SDF-based contact handling. Appendix E shows that the SDF-based representation has better collision handling than vanilla NeRF density fields. We also compare with NeRF-based simulation [269, 271] in Appendix E and ours achieves better performance.

7.4.2 Performance

In this section, we will show how our method can be used in photorealistic real-time gaming and physically-based simulation. A detailed profiling of our rendering and simulation modules can also be found in Appendix E.

Photorealistic Real-time Gaming. In addition to photorealistic rendering, our pipeline is fast, aiming to serve as a real-time neural-fields game engine. As shown in Figure 7.7, we have implemented an interactive game where players can control the ball’s motion using a keyboard and adjust camera viewing angles using a mouse. The background scene, excluding the green ball, is modeled by NeRF. The ball can have contact with the table and bulldozer. Players can also observe the bending of rays as they pass through the refractive glass ball. Supplementary material includes a recording of the real-time game. The game runs on a laptop with an NVIDIA GeForce RTX 2070 Max-Q GPU. Through our pipeline, game developers can seamlessly integrate animatable objects with photorealistic NeRFs.

Qualitative Results. Our method can simulate different types of dynamics with the support of Warp [291]. In Figure 7.6, the background fox is modeled as NeRF. (a) is a cube modeled by Neo-Hookean FEM mesh, where we can also see the changing light effect as it is twisted. (b) is a piece of cloth falling down to the fox. Our pipeline can handle the collision and light effects of such thin shells.

7.4.3 Applications

Our methods could be applied to many situations, where NeRFs can improve the realism of synthetic scenes.

Driving simulation is important for developing, training, and testing autonomous driving systems.

With the large number of images captured around roads, people can train NeRF for street views [244].

With our method, people can set up the driving simulation inside those Photorealistic NeRFs (see Figure 7.8 (a)) and insert synthetic vehicles [294, 295]. Such realistic virtual environments can help minimize the sim-to-real gap in self-driving cars.

Room layout design can help users design their homes and purchase furniture. After taking pictures and building a NeRF model for their room, customers can shop furniture [296, 297] virtually and design the room layout as shown in Figure 7.8 (b).

Virtual try-on using our methods can dynamically simulate the cloth on a human body captured by NeRF. In Figure 7.8 (c), we place a cloak on the human body and simulate how it swings in the wind.

Digital human applications are one of the key interests in VR/AR and the metaverse. With our methods, users can easily collect and build their virtual world by NeRF and then render their human-body model in that scene. This could be useful for movie making, webcasting, virtual performance, cyber-tourism, etc. Figure 7.8 (d) renders a futuristic ‘mercury man’ [298] jumping in the park using NeRF.

7.5 Conclusion

In summary, motivated by integrating NeRF into the traditional graphics pipeline, our hybrid rendering method can render dynamically changing meshes in a photo-realistic NeRF environment, without costly surface reconstruction. We also equip the resulting renderer with a simulator, making it suitable as a real-time NeRF-based game engine.



(a) Room layout design



(b) Driving simulator



(c) Virtual try-on



(d) Digital human

Figure 7.8: **Application of mixing NeRF with meshes.** Our method can be used for realistic driving simulation, room layout design, virtual try-on, and digital humans.

There are some limitations in this work. (1) The currently implemented renderer cannot cast shadows and illumination on NeRF points. Decomposing NeRFs can make the relighting more realistic. (2) Our renderer offers basic, essential functions; support for environment maps, UV maps, and image textures for higher rendering quality can be a natural extension. (3) Additional interfaces can also enable users to take advantage of more mature infrastructures if integrated into more widely-used platforms, e.g. Blender, Unreal, etc. (4) We can further improve the runtime performance and integrate the pipeline with larger-scale NeRFs.

Acknowledgements. This research is supported in part by Dr. Barry Mersky and Capital One E-Novate Endowed Professorships, and ARL Cooperative Agreement W911NF2120076. Yi-Ling would also like to thank the support from Meta Fellowship and Dr. Chunsheng Hu’s assistance in drawing the diagram.

Chapter 8: Physically Consistent Perception of Hand-Object Interactions with Differentiable Priors

8.1 Introduction

Hands serve as the primary means by which humans interact with the physical world as they are responsible for most of the intricate and dexterous object manipulation tasks. A good understanding of hand-object interaction helps with numerous learning-based tasks such as action recognition [299], robotic manipulation [300], user interaction [301], etc.

Estimating hand and object interaction presents several challenges. Firstly, the complex, deformable geometry and dynamics of hands present challenges in learning and optimization. Unlike rigid objects, hands are articulated with joints and possess high degrees of freedom. Their non-linearity and non-convexity further exacerbate the complexity of the task. Secondly, the presence of both self-occlusion and hand-object occlusion result in uncertainty when predicting hand poses during manipulation. Finally, the rich contacts between hands and objects in interaction scenes necessitate the need for a consistent estimation method for both the hand and the object to avoid penetration.

There are generally two categories of approaches for estimating hand and object poses. The first approach is purely learning-based, where the network architectures, loss functions,

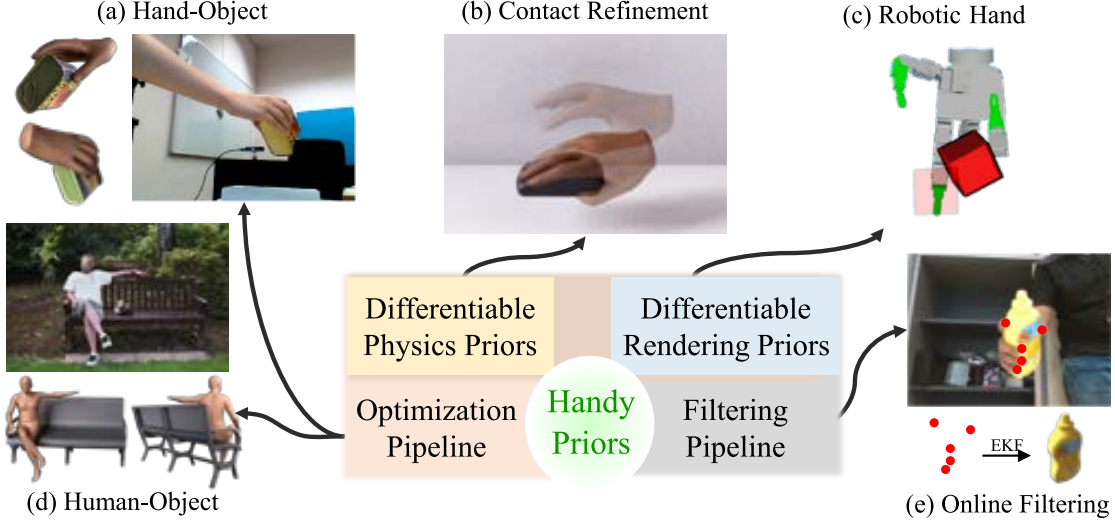


Figure 8.1: **HANDYPRIORS is a modular block application in multiple tasks.** The estimation of hand-object interaction can be achieved by (a) optimization-based refinement or (e) filtering with Extended Kalman Filter (EKF). Moreover, the differentiable contact module can be used to perform (b) pose refinement given contact status. Our differentiable priors can also be used in pose estimation for scenes with (c) robot hands or (d) human bodies. and training strategies are carefully designed to train a generalizable mapping from images to

hand and object configurations [302, 303]. Although neural networks and large training datasets can mitigate the difficulties posed by high degrees of freedom and non-convexity, they cannot guarantee interpretability nor correctness. The second category of approaches is optimization-based [304, 305], which involves defining and optimizing heuristic target functions to improve physical feasibility and reduce penetrations. Although the optimization process may be slower, per-frame iterations can improve the accuracy of the results.

In this chapter, we leverage both learning and optimization techniques. Our method utilizes pre-trained networks to provide an initial estimate of poses, and then uses differentiable self-supervised terms to further refine the estimation by correcting errors from pixel-level misalignment and non-physical penetration/oscillation. Inspired by the success of self-supervised learning in large-scale vision and language models [306, 307], we adopt similar techniques to estimate hand-object interactions. Specifically, we propose *supervising the pose prediction by comparing it*

with subsequent video frames, similar to the way in which text generation can be trained using sequence completion.

To enable backpropagation of gradients from image space, we develop *fully-differentiable operators that simulate and render predicted poses*. This closed-loop optimization approach improves the accuracy by making the estimation resemble more of the input image. We also *incorporate several regularization terms to minimize motion oscillation and collision in predicted sequences*.

While the optimization process improves the accuracy, it can be time-consuming. To address this issue, we present an alternative filtering technique based on the Kalman filter [308] to further *utilize our differentiable dynamics and observation modules*. This approach allows the user to make a trade-off between efficiency and accuracy depending on whether they choose to perform online tracking or offline optimization. Employing this integrated differentiable rendering and simulation pipeline, we also apply our pose estimation method to human and synthetic robot hand scenes, demonstrating the generalizability and robustness of our method. Figure 8.1 summarizes several applications of this approach.

In summary, the key contributions of this work are:

- An **integrated differentiable rendering** (Sec. 8.3.2) and **simulation** (Sec. 8.3.3) pipeline to estimate hand-object interaction.
- An **offline optimization** (Sec. 8.3.4) and an **online tracking** (Sec. 8.3.5) process that can predict poses and contact information, providing the user with options to balance between accuracy and efficiency.
- **Experiments that generalize our priors to applications**, e.g. robotic manipulation (Sec. 8.4.1),

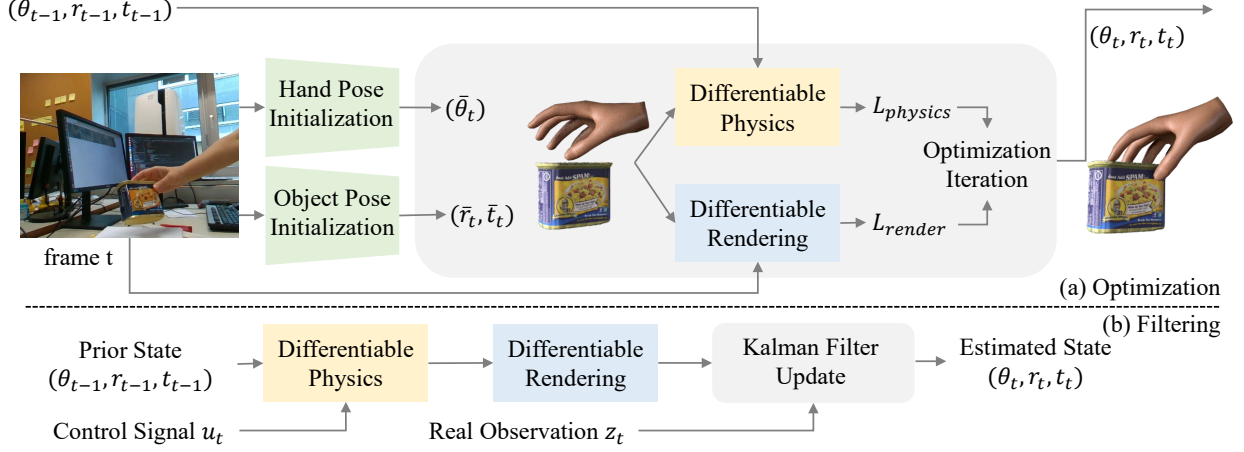


Figure 8.2: **An overview of our optimization and filtering pipelines.** We provide two alternatives for utilizing the differentiable priors. Given the image and the estimation from previous frames, (a) the optimization pipeline first initializes the poses with pre-trained networks and then minimizes the rendering and physics losses from the differentiable operators; (b) the filtering pipeline can take some simple observations and use Extended Kalman Filter (EKF) to update the state estimation. EKF requires differentiable physics and rendering to model the system and runs much faster than the optimization pipeline.

contact refinement (Sec. 8.4.4), and human bodies (Sec. 8.4.5).

In our experiments, our method achieves comparable or better pose estimation results in hand-object-interaction datasets [309, 310]. Additionally, we show that the contact information predicted by our method effectively refines pose estimation. Our ablation study compares the differences between the optimization and tracking results, further validating the contributions of different losses.

8.2 Related Works

8.3 Methods

Hand and object pose estimation from monocular images [311, 312, 313, 314] is a fundamental problem in computer vision. Many recent works predict 6 DoF object poses assuming known

rigid mesh models [315, 316, 317]. For human hands, MANO [318] proposes a parametric model to represent the hand geometry using different shape and pose parameters. Numerous learning-based approaches focus on predicting 3D joint locations from RGB(D) inputs [319, 320, 321, 322, 323, 324, 325]. Others [305, 326, 327, 328] then regress to the MANO pose and shape parameters. There are also approaches directly learning a mapping between visual inputs and hand poses from annotated datasets [329, 330, 331, 332]. These methods achieved remarkable success on hand-only datasets but had poor performance when hands were holding objects. There are also optimization-based approaches that optimize hand parameters based on 2D key points [304, 305, 333] or segmentations [309]. They tend to have better accuracy but are much slower due to the iterative optimization process. We propose a tracking-based pipeline with higher efficiency. Our differentiable priors are *more general* and can be applied to a *wider range* of applications.

Hand object interaction has been garnering progressively more attention. Treating hand and object pose estimation separately results in inaccurate estimation for both. With an increasing number of hand-object datasets being created [302, 309, 334, 335, 336, 337], many recent works have started to focus on jointly tracking hands and objects. However, simultaneously estimating both is challenging due to occlusions and depth ambiguity. [303] designed a joint learning framework based on contextual reasoning between hand and object representations. [338] proposed an ordinal relation loss to correct depth misalignment between hand and object. [339] incorporated a feature injection mechanism that better handles occlusion by integrating hand information into the obscured areas. These methods improve the accuracy of pose estimation but pay little attention to the physical interaction between hand and objects. Other works aim to predict

physically plausible hand-object poses. To achieve this goal, interaction constraints are applied to encourage contact and minimize interpenetration. [304, 305] proposed to use signed distance functions (SDF) to model contact between hand and object, [340] design a deep network to estimate contact areas and a virtual capsule technique to simulate soft hand tissue deformation. [341] apply motion and force vectors to reconstruct interactions. Compared to these methods, our approach is more general and can be used in *contact-based refinement and pose estimation in both human-hand and non-human-hand scenarios*.

Differentiable Priors. Differentiable rendering [86, 135, 342] and physics [158, 343, 344] have been used to solve inverse problems for their efficiency in optimization. For hand grasping, [345] leverages a physics simulator to evaluate dynamic interactions between the hand and the object. [346] uses differentiable physics to generate physically realistic and stable grasps. There are also recent works that use differentiable rendering to optimize human [347] or object [348] poses. However, they do not consider their physical interactions. [349] utilize a differentiable articulated body simulator to learn and smoothen human-pose estimation. In contrast, our method leverages rendering priors for each frame, thereby significantly accelerating pose estimation in a video.

8.3.1 Problem Setting

Given a sequence of RGB hand-object-interaction images and the object’s 3D model, we propose two pipelines to estimate the configuration of the hand and object in each frame. Figure 8.2 shows an overview of our method.

In particular, we represent the hand using the MANO [318] parameterization, which describes

a hand by its pose $\theta \in \mathbb{R}^{45+6}$ (representing the rotations of 15 hand joints and a 6 DoF global transformation) and shape $\beta \in \mathbb{R}^{10}$ (a statistical model) parameters. The 778 vertices $\mathbf{V}_{hand} \in \mathbb{R}^{778 \times 3}$ of the hand mesh can be computed from θ, β using a learned linear model $\mathbf{V}_{hand} = m(\theta, \beta)$.

The object pose has 6 degrees of freedom, which contains its translation $\mathbf{t} \in \mathbb{R}^3$ and rotation (represented by XYZ Euler angles) $\mathbf{r} \in \mathbb{R}^3$.

8.3.2 Differentiable Rendering Priors

The first prior of our approach is differentiable rendering, which helps align the estimated 3D model with the 2D image. Instance segmentation is the key information we rely on to optimize hand and object poses. We use a segmentation detector [350] trained on the COCO dataset [351] to extract hand and object masks. Our experiment shows that although the image segmentation detector, *PointRend* [350], was not trained on the YCB objects [352], it can correctly predict object masks. However, it fails to adequately segment hands as fingertips are harder to detect. Thus, we further trained *PointRend* to obtain better segmentations for the hand and hand-held objects.

For each frame, we utilize a differentiable renderer *SoftRas* [342] that rasterizes and projects 3D hand and object models to 2D, then we optimize hand parameters θ and object parameters $\mathbf{r}, \mathbf{t}$ by fitting the projected vertices onto the hand and object masks. Next, we describe loss terms related to the rendering part.

RGB image loss. Since hand texture varies greatly from person to person, we only render the RGB images for objects. We then apply L2 loss between the rendered image I_r and the input

RGB image I_{in} cropped by the detected object mask M_{obj} . The image loss term can be expressed as follow:

$$L_{img}(\mathbf{r}, \mathbf{t}) = \|(1 - M_{hand})I_r - M_{obj}I_{in}\|_2^2 \quad (8.3.2.1)$$

Mask loss. Compared to the RGB image loss, masks have fewer features but are not sensitive to lighting and texture, hence the loss based on masks is more robust and general. We use a differentiable renderer to render the silhouette of hands and objects together. We then apply L2 loss between the rendered mask M_r and the detected mask M_d ,

$$L_{mask}(\theta, \mathbf{r}, \mathbf{t}) = \|M_r - M_d\|_2^2 \quad (8.3.2.2)$$

8.3.3 Differentiable Physics Priors

Besides the rendering priors defined on the 2D image spaces, we also add more physics-related priors in the 3D and temporal domain to help us regularize the hand-object understanding.

Contact loss. During manipulation, we observe that most of the contacting vertices between hand and objects do not have sudden relative sliding. Based on this observation, we first utilize a GPU-based differentiable contact module to infer the signed distance function $sdf(\mathbf{V}_{hand})$ from hand vertices to the object. Then, we find the contact points $\mathbf{v}_t$ where the SDF values are smaller than 0.1. The relative sliding between t and $t - 1$ is,

$$L_{sliding}(\theta, \mathbf{r}, \mathbf{t}) = \|(\mathbf{v}_t - \mathbf{t}_t)(\mathbf{r}_t^{-1})^\top - (\mathbf{v}_{t-1} - \mathbf{t}_{t-1})(\mathbf{r}_{t-1}^{-1})^\top\|$$

Moreover, given the SDF results, we reduce penetration by penalizing the negative SDF

values.

$$L_{penetration}(\theta, \mathbf{r}, \mathbf{t}) = \min(0, -sdf(\mathbf{V}_{hand})) \quad (8.3.3.1)$$

Smoothness loss. To enhance the continuity of hand and object across time, we penalize sudden changes of object and hand in two consecutive frames. For objects, we apply L2 loss between object vertices in the current frame $\mathbf{V}_t$ and previous frame $\mathbf{V}_{t-1}$. For hands, we compute the L2 loss between hand poses in two frames θ_t, θ_{t-1} ,

$$L_{con} = \|\mathbf{V}_t - \mathbf{V}_{t-1}\|_2^2 + \|\theta_t - \theta_{t-1}\|_2^2. \quad (8.3.3.2)$$

During the experiment, we also observed that objects usually have jittering for several frames. Thus, we add a smoothness term to punish the oscillation in object velocity,

$$L_{smo} = \|(\mathbf{V}_t - \mathbf{V}_{t-1}) - (\mathbf{V}_{t-1} - \mathbf{V}_{t-2})\|_2^2. \quad (8.3.3.3)$$

8.3.4 Optimization-based Refinement

In summary, the losses of rendering priors are

$$L_{render} = \lambda_1 L_{image} + \lambda_2 L_{mask}, \quad (8.3.4.1)$$

and the physics-inspired losses are

$$L_{physics} = \lambda_3 L_{sliding} + \lambda_4 L_{penetration} + \lambda_5 L_{con} + \lambda_6 L_{smo},$$

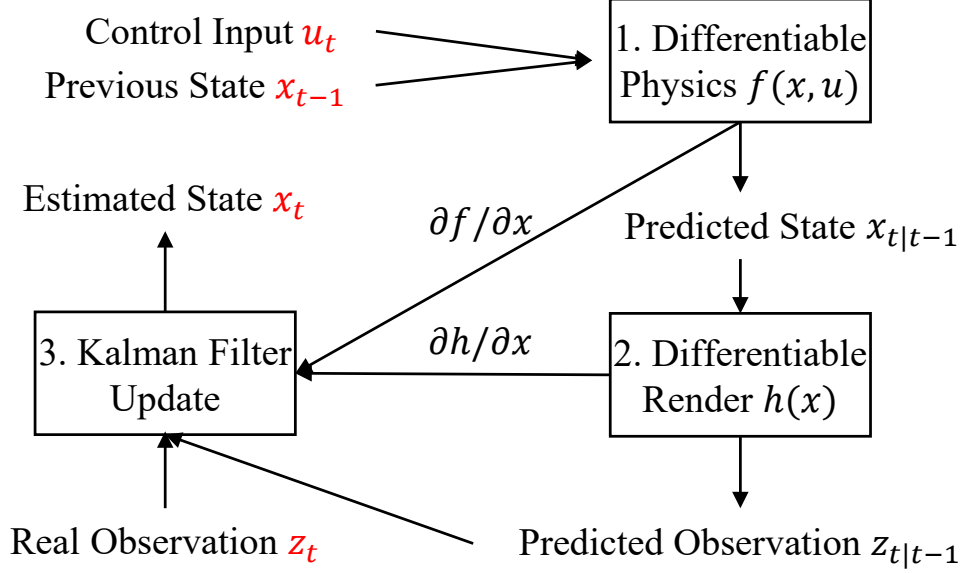


Figure 8.3: **Diagram of the EKF filtering process.** With (1) the differentiable physics f and (2) rendering h modules to compute the Jacobians, (3) the extended Kalman Filter (EKF) can perform state estimation given the control signal, observation, and previously estimated states.

Table 8.1: **Ablation study.** We run ablation studies for optimization (left) and filtering (right) pipelines. For optimization, 2D and 3D errors (*cm*) decrease with gradually added rendering $\mathcal{L}_{render}$ and physics loss $\mathcal{L}_{physics}$. Filtering (0.03 s) is faster than the optimization (e.g. ours take 5 s and Homan [305] takes 9 s) but sacrifices accuracy.

Optimization	2D Error	3D Error	Runtime (s)	Filtering	2D Error	3D Error	Runtime (s)
Initialization	1.03	52.5	0.072	$\mathcal{O}_{hand}$	44.59	5.57	0.024
+ $\mathcal{L}_{render}$	0.77	1.54	2.29	$\mathcal{O}_{object}$	19.00	11.24	0.027
+ $\mathcal{L}_{physics}$	0.43	1.10	5.83	$\mathcal{O}_{hand+object}$	8.84	7.41	0.029

where λ 's are the weights for the loss terms. With these differentiable loss terms, we optimize the pose of the hand θ and object $\mathbf{r}, \mathbf{t}$ using a gradient-based optimization method such as Adam [353].

8.3.5 Filtering-based Tracking

Besides the optimization-based process, we can also use the differentiable priors in a more light-weight (faster but less accurate) pipeline (Fig. 8.3). Multiple filtering algorithms (e.g.

Extended Kalman Filter (EKF) [308]) can be integrated with this method for fast tracking. EKF can give a state estimation in a dynamic system

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t) + \epsilon_s, \mathbf{z}_t = h(\mathbf{x}_t) + \epsilon_o, \quad (8.3.5.1)$$

where $\mathbf{x}$ is the state, $\mathbf{u}$ is the control, $\mathbf{z}$ is the observation, ϵ_s and ϵ_o are noises, $f(\cdot), h(\cdot)$ are *differentiable* functions for states and observation, respectively. Using the EKF, we can compute an estimation $\hat{\mathbf{x}}_t$ for the hidden state $\mathbf{x}_t$ given the observation $\mathbf{z}_t$ and control $\mathbf{u}_t$.

The states of the hand-object system are $\mathbf{x}_t = (\theta_t, \mathbf{r}_t, \mathbf{t}_t)$. Using those pose parameters (along with the known object model and shape parameters β), we can reconstruct the entire scene. For most of the cases in real life and nearly all of the cases in existing datasets, hands can be modelled as actuators whereas objects are primarily passive. Therefore, the control input for the system is the hand pose $\mathbf{u}_t = \theta$.

Since the differentiable renderer can output various quantities (2D keypoints, bounding box, full image set), the observations we can choose are also flexible. For instance, the 2D positions of the fingertips can serve as the only feedback supervision. Then the observation function is,

$$h(\mathbf{x}_t) = S_{tips} \cdot \mathbf{V}_{hand}(\theta_t), \quad (8.3.5.2)$$

where $S_{tips} \in \mathbb{R}^{5 \times 778}$ is a one-hot encoding matrix that selects fingertips out of the hand.

The more challenging part is to construct the dynamics function $f(\cdot)$. A direct solution is to use a differentiable full dynamics model to describe the system. However, the actuation signals of the hand (like muscle contraction) are unavailable from the MANO parameter. We thus choose

to use simpler and more robust quasistatic dynamics. If the hand makes contact with an object on vertices $\mathbf{v}_{t-1} = S_{contact} \cdot \mathbf{V}_{hand}(\theta_{t-1})$ at time step $t - 1$, we assume the relative sliding on the vertices from $\mathbf{v}_{t-1}$ to $\mathbf{v}_t \in \mathbb{R}^{n_{contact} \times 3}$ is small. The rotation $\mathbf{R}$ and translation $\mathbf{T}$ are estimated as,

$$\mathbf{U}\Sigma\mathbf{V}^\dagger = svd(\tilde{\mathbf{v}}_{t-1}^\top \tilde{\mathbf{v}}_t) \quad (8.3.5.3)$$

$$\mathbf{R}_t = (\mathbf{U}\mathbf{V}^\dagger)^\top \mathbf{R}_{t-1}$$

$$\mathbf{T}_t = \bar{\mathbf{v}}_t - \bar{\mathbf{v}}_{t-1}(\mathbf{U}\mathbf{V}^\dagger) + \mathbf{T}_{t-1}$$

where $\tilde{\mathbf{v}} = \mathbf{v} - \bar{\mathbf{v}}$ is the contact vertices subtracting their centroids $\bar{\mathbf{v}}$, $svd(\cdot)$ is the singular value decomposition.

With the differentiable observation $h(\cdot)$ (Eq. 8.3.5.2) and dynamics $f(\cdot)$ (Eq. 8.3.5.3), we can then use the EKF to track and estimate the states $\mathbf{x}_t = (\theta_t, \mathbf{r}_t, \mathbf{t}_t)$.

8.4 Experiments

In this section, we apply our differentiable priors to five scenarios: (a) object pose estimation for robotic manipulation, (b) pose estimation, (c) filtering-based tracking, (d) contact refinement, and (e) human-object pose estimation. Our method can achieve comparable or better results in a wide range of tasks.

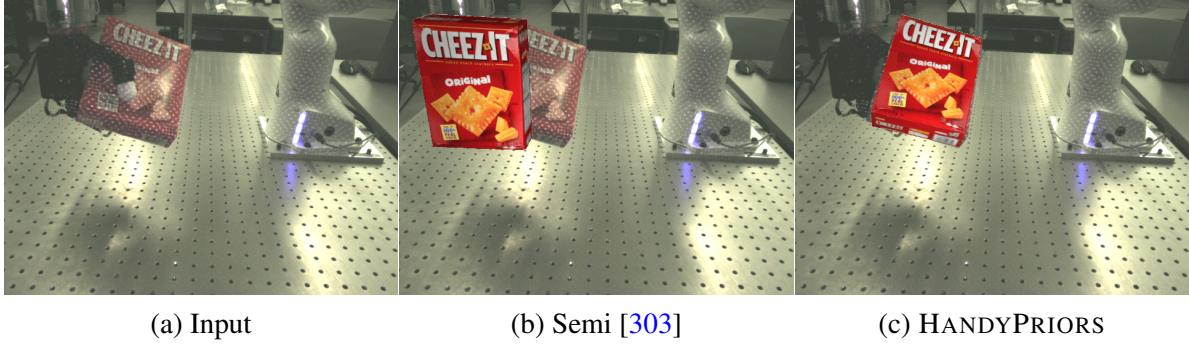


Figure 8.4: **Real-world pose estimation.** (a) We record a robotic hand manipulating a cracker box in the real world using a calibrated camera. With (b) initialization using [303]. We observe that (c) our method can recover the transformation of the box.

8.4.1 Real-world Robotic Hand and Object Interaction

Our method can generalize to a wide range of scenarios, including robotic hands and object interaction. To demonstrate this, we designed a robotic hand manipulation task. We first record sequences of RGB images where an Allegro robotic hand interacts with a YCB object [352] using the calibrated Intel RealSense camera and record the corresponding hand pose parameters. We then predict the initial object pose using only the RGB image with [303]. Given the robotic hand configuration calculated using the hand pose parameter and segmentation masks predicted by Pointrend [350]. We then use our pipeline to optimize the object pose r, t . Figure 8.4 shows the optimization result of our method. The results demonstrate that our method is capable of generating accurate object poses even with robotic hand interaction scenes.

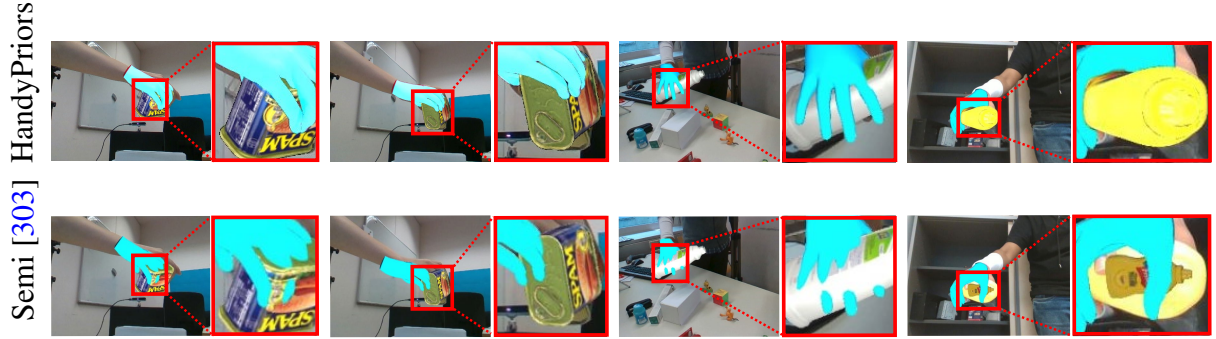


Figure 8.5: **Pose Estimation Results.** We compare the hand and object pose estimation results with other methods and visualize the 3D meshes projected on the original image. Our method yields reasonable joint estimation for the interaction scenes.

Table 8.2: **Evaluation of hand pose estimation.** We measure the joint error (*cm*), mesh error (*cm*), and joint AUC on the DexYCB dataset [336]. PA is Procrustes Alignment.

Methods	Joint (PA)	Joint	Mesh (PA)	Mesh	Joints AUC
Artiboost [338]	0.65	1.12	0.69	1.21	0.870
Homan [305]	1.84	4.41	1.80	4.34	0.644
HandOccNet [339]	0.62	1.16	0.66	1.20	0.877
HANDYPRIORS (ours)	0.52	0.81	0.51	0.89	0.896

Table 8.3: **Quantitative evaluation of object pose estimation.** Our approach is compared with Semi [303], Artiboost [338], Homan [305] on the HO3D [309] and DexYCB Dataset [336]. We report the 2D (*cm*) and 3D (*cm*) errors.

Method	2D Error (HO3D)	3D Error (HO3D)	2D Error (DexYCB)	3D Error (DexYCB)
Artiboost [338]	6.41	10.83	3.01	6.78
Homan [305]	4.60	41.33	15.21	95.07
Semi [303]	2.06	26.35	9.20	47.77
HANDYPRIORS (ours)	1.67	3.22	0.96	3.79

8.4.2 Pose Estimation

8.4.2.1 Datasets and Metrics

Datasets. HO3D dataset [309] is a real hand-object interaction dataset that contains approximately 80K images from more than 65 sequences. It uses 10 models from the YCB dataset [352]. We report the performance of all of the methods on the evaluation split of the dataset. Dex-YCB dataset [336] consists of 582K images over 1,000 sequences of 10 subjects grasping 20 different YCB objects [352] from 8 camera views. In this work, we report the performance of our method on the S_0 test split of the dataset. Following [338], we filter out frames that contain the left hand or where the hand is not in contact with the object. **Metrics.** In Table 8.3, we measure the 2D error by projecting the object vertices onto the image plane and computing the Chamfer distance between the ground truth vertices and predicted vertices. The 3D error is the Chamfer distance in the original 3D space. We also report the collision between the hand and the object. In Table 8.2, we compute the joint error, mesh error, and joint AUC (Area Under the ROC Curve). For joint and mesh error, we report results before and after Procrustes alignment, where Procrustes alignment changes the global rotation, translation, and scale of the estimated pose to match the ground truth.

8.4.2.2 Optimization-based Refinement

We compare our optimization-based method described in Section 8.3.4 with Semi [303], Artiboost [338], Homan [305] for the pose estimation task. In the HO3D dataset, we test three objects from the evaluation split: Mustard Bottle (SM11), Meat Can (MPM10-15), and Bleach Cleanser (SB11, 13). Table 8.3 shows that our method outperforms alternative methods for estimating the pose of the objects. Our method also achieves comparable results in terms of collision metrics. Table 8.2 shows the hand prediction results compared to Artiboost [338], Homan [305], and HandOccNet [339]. Our method also improves over the initialization from [339] and achieves the best accuracy in all metrics. Figure 8.5 shows some qualitative results of the predictions. Compared to other methods, our optimized results are better aligned with the image and have more reasonable relative positions between hands and objects.

8.4.2.3 Ablation Study

In this part, we study the influence of different loss terms on the estimation results. All the experiments are run in the evaluation set SM1 from the HO3D dataset [309]. Table 8.1 presents the quantitative results. Starting from the initialization by the pre-trained network [303], adding rendering loss $\mathcal{L}_{rendering}$ drastically improves the performance, especially with regards to the 3D error as it can adjust the translation errors in the depth direction. Moreover, adding the physics loss term $\mathcal{L}_{physics}$ improves the results further by reducing penetration and oscillation in the predicted sequences.

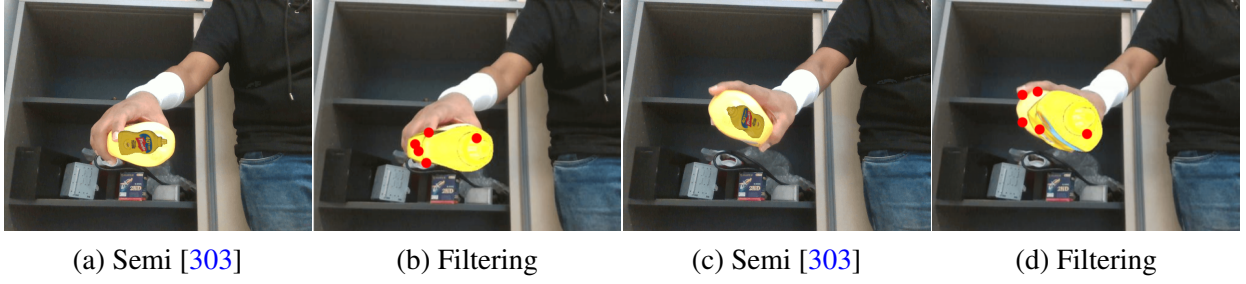


Figure 8.6: **Filtering results.** We show that the filtering pipeline can estimate the object pose with just fingertips (red points) as input. It can give robust and continuous estimation even when the purely learning-based method (a, c) fails.

8.4.3 Filtering-based tracking

We apply the tracking algorithm described in Section 8.3.5 to perform pose estimation. We run on the SM1 sequence and qualitative results are shown in Figure 8.6. Our filtering pipeline (b) yields adequate estimation results of the objects given only the fingertips as inputs (red dots in the figures). Meanwhile, the (a) learning-based method [303] would have huge errors in object pose estimation.

We also conduct quantitative experiments in Table 8.1 where different observations are fed into the filter. $\mathcal{O}_{hand}$ use the predicted 3D positions of fingertips from as observation; $\mathcal{O}_{object}$ takes the predicted object center as observations; $\mathcal{O}_{hand+object}$ uses both the aforementioned hand and object information as an observation. The results show that with more observation available, the tracking results tend to be more accurate. The tracking method is much faster than the optimization pipeline; however, the accuracy decreases since less information is provided to it.

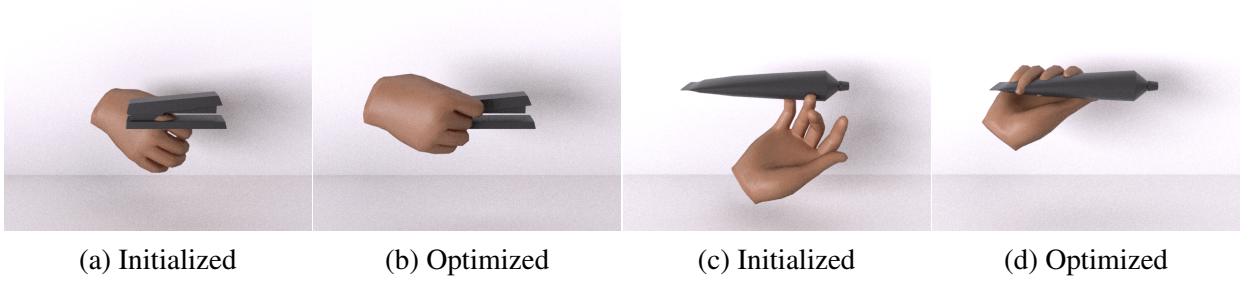


Figure 8.7: **Qualitative results of contact-based pose refinement.** Given randomized initial poses (a,c) and desired contact regions, our method can refine poses using a differentiable contact model & gradient-based optimization (b,d).

8.4.4 Contact Refinement

The physics module can output contact information during prediction. Given the pose of the hand and the object, our GPU-based differentiable Warp kernel [291] can output the signed distance function s and the normal direction $\mathbf{n}$ for each hand vertex $\{\mathbf{n}_i, s_i\}_{i < n_{hand}} = \text{contact}(\theta, \mathbf{r}, \mathbf{t})$. Following [340], we define a contact value $CO_i = \min(\frac{1}{s_i}, 1)$ for each vertex. When a vertex is within 1mm ($s_i < 1$) of the object, $CO_i = 1$ then means this point is in-contact.

Given randomly initialized hand poses, our method refers to the ground truth contact information provided by ContactPose datasets [354] as the target. Since our $\text{contact}(\cdot)$ module is also differentiable, we can then optimize the poses to match the target contact status $\{CO_i, \mathbf{n}_i\}$ with gradient-based optimization and achieve comparable results as [340] shown in Figure 8.4.

Figure 8.7 shows examples of the optimization process. (a) and (c) are the input randomized hand, (b) and (d) are the optimized hand-object interaction. The results are similar to how people handle those objects in daily life.

Table 8.4: **Error after contact-based pose refinement.** Our differentiable contact module achieves comparable results as ContactOpt [340].

Method	hand joints	hand verts	object verts
ContactOpt	0.0143	0.0137	0.0367
ours	0.0143	0.0136	0.0364

Table 8.5: **Quantitative evaluation on human object pose estimation** Our method can achieve higher 2D IoU and lower collision error (cm^3) compared to PHOSA [348].

Method	human IoU	object IoU	Collision
PHOSA	0.45	0.51	6818.18
$+\mathcal{L}_{render}$	0.64	0.59	8742.63
$+\mathcal{L}_{physics}$	0.68	0.64	666.67

8.4.5 Generalization to Human

Our proposed differentiable priors can also be applied to human object pose estimation. With RGB images from the COCO2017 dataset [351], we use [348] to initialize both human and object poses, and then use [350] to predict the instance segmentation mask for both the human and objects. Subsequently, we replaced the MANO hand model [318] in our pipeline by SMPL human body model [298] which describes the human body also by pose $\theta \in \mathbb{R}^{72}$ and shape $\beta \in \mathbb{R}^{10}$ parameters. Following the same set up as hand-object optimization, we refine human parameters $(\theta, \beta, T_{human})$ and object parameters (r, t) through our pipeline using both differentiable rendering and physics priors. Our experiment shows that our method can generate accurate human-object pose estimation. As ground truth poses for human and object are unavailable, we report the intersection of union(IoU) and the interpenetration volume between human and object in Table 8.5. Figure 8.8 shows the qualitative result of our method and the contribution of each component of the pipeline.

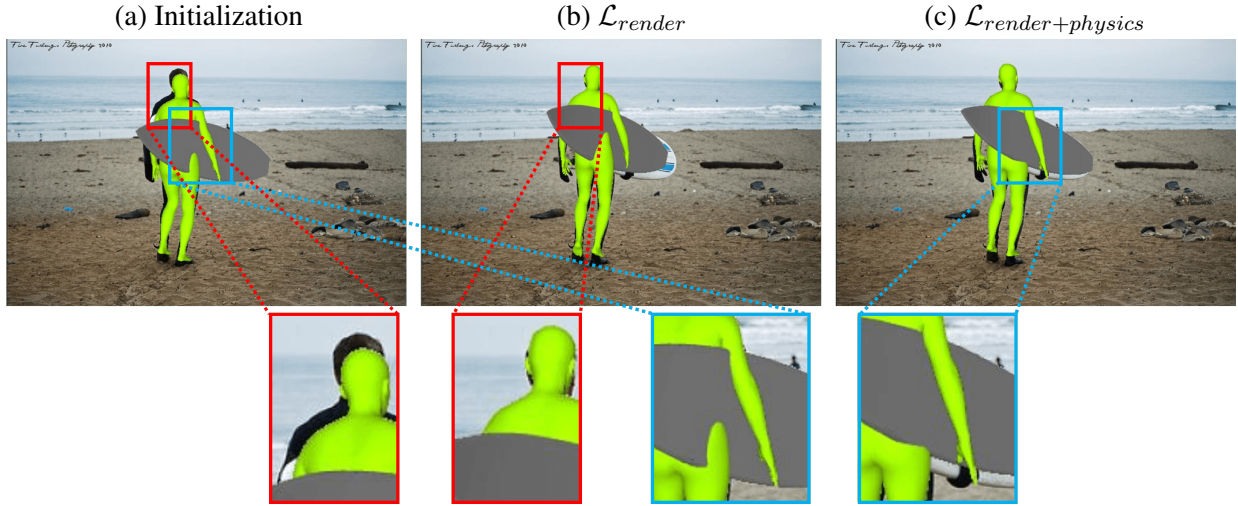


Figure 8.8: **Human Object Pose Refinement.** Qualitative comparison between (a) initialization using [348], (b) refinement using only differentiable rendering, and (c) refinement using both differentiable rendering and physics simulator. The rendering loss helps to align the image, and the physics loss can mitigate penetrations.

8.5 Conclusion

In this chapter, we present HANDYPRIORS, a framework of differentiable rendering and physics priors for pose estimation in hand-object-interaction scenes. Two alternatives are presented to match different use cases: the optimization-based pose refinement which focuses on the prediction accuracy, and the filtering-based tracking which focuses on the runtime speed. Our proposed rendering and physics-related operators enable closed-loop optimization and tracking with self-supervision from the input RGB image sequences. Experiments show that our method yields comparable or better results in the pose estimation and contact refinement task due to our unified and general priors. We also demonstrate that our method can be applied to robotic hand manipulation and human pose estimation tasks.

There could be several future follow-ups to this work. First, the optimization pipeline can fail if the initialization is poor. This problem could be alleviated by adding bounding boxes

or key-point related terms to the differentiable rendering module. Second, the performance of the optimization could be improved through a faster differentiable rendering module, such as Nvdiffrast [355]. Third, the current filtering pipeline based on EKF could potentially output the uncertainty of the prediction. It would be promising to incorporate our pipeline with probabilistic prediction. Fourth, while it currently supports human hands, it could be adapted for robotic arms with different configurations by changing the kinematic structures.

Chapter 9: Other Visual-Dynamical Systems

9.1 Navigation and Collision Avoidance Using Commodity Sensors

Mobile robots are currently deployed in a wide range of scenarios, including warehouses, airports, malls, and offices [356]. In these indoor and outdoor spaces, robots are used to perform routine tasks such as delivering goods or guiding customers. The ability to perform reliable navigation and avoid collisions with pedestrians and other obstacles is important to perform these tasks.

There are extensive works on robot navigation and collision-avoidance in dynamic scenes [357, 358]. These include model-based techniques based on velocity obstacles [359] and dynamic constraints [357]. However, these methods assume accurate detection or representation of obstacles in the environment [357, 359]. Other techniques have been proposed for dynamic scenes [358, 360], but their performances vary in different environments and none of them are robust enough for all general scenarios [361].

Recently, a number of learning-based approaches have been proposed to perform robot navigation and collision avoidance [362, 363, 364]. These algorithms can handle sensor noise in many scenarios. However, it is challenging to predict the performance in new or unknown scenes with those learning-based models because their performance is governed by the training data. Some model-based navigation algorithms [365, 366] use probabilistic techniques to handle

sensor errors, but they are not robust in terms of handling partial observations.

Main Results: We present a novel hybrid scheme for safe navigation of a mobile robot among pedestrians and other obstacles. Our approach takes advantage of learning-based methods to handle sensor data (perception) and a model-based formulation (planning) to compute safe velocities for collision avoidance. Instead of developing an end-to-end learning pipeline, we use separate components for perception and planning to provide guarantees of collision avoidance. The novel components of our approach include:

- We integrate a segmentation and an optical flow estimation network to detect obstacles. The state-of-the-art pretrained models helps robots perceive the environment accurately in real time.
- We present a modified velocity-obstacle algorithm to perform local navigation and collision avoidance based on partial observations. We highlight improvements over prior velocity obstacle methods and show that our approach is more robust.
- We present bounds for our probabilistic collision avoidance. These bounds provide better guarantee and interpretability than learning-based navigation schemes.

We have tested our algorithm on a Turtlebot robot with commodity sensors, including an Orbecc Astra RGB camera (around \$100) and a 2-D Hokuyo Lidar (costing around \$1000). We have evaluated OF-VO in simulated environments with multiple pedestrians, including high-density cases with more than 10 pedestrians in $5 \times 5 \text{ m}^2$ area. We used different methods to simulate pedestrian movements, including socially acceptable collision avoidance [367, 368]. Moreover, we tested the algorithm in our laboratory with pedestrians close to the robot and moving in different patterns. Results will be shown in the video. We compared our approach

with prior model-based methods like DWA [357] and deep-reinforcement learning algorithms like [369]. We observe 2 – 5X improvement in success rate (and reliability).

This is a collaboration work led by Jing Liang. More details can be found in the full paper [370].

9.2 Physics Augmented Continuum Neural Radiance Fields

Inferring the geometric and physical properties of an object directly from visual observations is a long-standing challenge in computer vision and artificial intelligence. Current machine vision systems are unable to disentangle the geometric structure of the scene, the dynamics of moving objects, and the mechanisms underlying the imaging process – an innate cognitive process in human perception. For example, by merely watching someone kneading and rolling dough, we are able to disentangle the dough from background clutter, form a predictive model of its dynamics, and estimate physical properties, such as its consistency to be able to replicate the recipe. There exists a large body of work on inferring the geometric (*extrinsic*) structure of the world from multiple images (e.g., structure-from-motion [371]). This has been bolstered by recent approaches leveraging differentiable rendering pipelines [372] and neural scene representations [373], unlocking a new level of performance and visual realism. On the other hand, approaches to extract the physical (*intrinsic*) properties (e.g., mass, friction, viscosity) from images are yet nascent [12, 211, 374, 375] – all assume full knowledge of the geometric structure of the scene, thereby limiting their applicability.

The key question we ask in this work is “*can we recover both the geometric structure and the physical properties of a wide range of objects from multi-view video sequences*”? This

dispenses with all of the assumptions made by state-of-the-art approaches to video-based system identification (known geometries in [211] and additionally rendering configurations in [12]). Additionally, the best performing approaches to recover geometries (but not physical properties) of dynamic objects in videos include variants of neural radiance fields (NeRF) [79], such as [104, 105, 106]. However, all such neural representations of dynamic scenes need to learn object dynamics from scratch, requiring a significant amount of data to do so, while also being uninterpretable. We instead employ a differentiable physics simulator as a more prescriptive, data-efficient, and generalizable dynamics model; enabling parameter estimation solely from videos.

Our approach—Physics Augmented Continuum Neural Radiance Fields (PAC-NeRF)—is a novel system identification technique that assumes nothing about the geometric structure of a system. PAC-NeRF is extremely general – operating on deformable solids, granular media, plastics, and Newtonian/non-Newtonian fluids. PAC-NeRF brings together the best of both worlds; differentiable physics and neural radiance fields for dynamic scenes. By augmenting a NeRF with a differentiable continuum dynamics model, we obtain a unified model that estimates object geometries *and* their physical properties in a single framework.

Specifically, a PAC-NeRF $\mathcal{F}$ is a NeRF, comprising a volume density field and a color field, coupled with a velocity field $\mathbf{v}$ that admits the continuum conservation law: $\frac{D\mathcal{F}}{Dt} = 0$ [376]. In conjunction with a hybrid Eulerian-Lagrangian formulation, this allows us to advect geometry and appearance attributes to all frames in a video sequence, enabling the specification of a reconstruction error in the image space. This error term is minimized by gradient-based optimization, leveraging the differentiability of the entire computation graph, and enables system identification over a wide range of physical systems, where neither the geometry nor the rendering configurations are known. Our hybrid representation considerably speeds up the original MLP-based NeRF

by efficient voxel discretization [377], and also conveniently handles collisions in continuum simulations, following the MPM pipeline [378]. The joint differentiable rendering-simulation pipeline with a unified Eulerian-Lagrangian conversion is highly optimized for high-performance computing on GPU.

In summary, we make the following contributions.

- We propose **PAC-NeRF** – a dynamic neural radiance field that satisfies the continuum conservation law.
- We introduce a *hybrid Eulerian-Lagrangian representation*, seamlessly blending the Eulerian nature of NeRF with MPM’s Lagrangian particle dynamics.
- Our framework estimates *both* the geometric structure and physical parameters of a wide variety of complex systems, including elastic materials, plasticine, sand, and Newtonian/non-Newtonian fluids, outperforming state-of-the-art approaches by up to **two orders of magnitude**.

This is a collaboration work led by Xuan Li. More details can be found in the full paper [272].

Part IV

Conclusion

Chapter 10: Conclusion

10.1 Summary of Results

In summary, my research focuses on building differentiable, high-fidelity, and feature-rich simulation pipelines and applying them to control, learning, and system identification tasks. My contributions can be categorized as follows,

Differentiable Simulation for Cloth and Rigid Bodies [4]. In this project, we develop a scalable differentiable simulator for rigid bodies and cloth. We evaluate the scalability and generality of the presented approach experimentally. Controlled experiments that vary the number of objects in the scene and their relative scales demonstrate that our approach is dramatically more efficient than state-of-the-art differentiable physics frameworks. Code is available on our project page: <https://gamma.umd.edu/researchdirections/mlphysics/diffsim>

Differentiable Articulated Body Simulation [156]. In this project, we design a differentiable articulated body simulation method that runs 10x faster with 1% of the memory consumption compared to differentiation based on autodiff tools. As an application of differentiable dynamics for articulated systems, we show that reinforcement learning (RL) can benefit from the knowledge of gradients in two ways. First, it can make use of the gradients computed by the simulation to generate extra samples using first-order approximation. Second, during the policy learning phase,

differentiable physics enables us to perform a one-step rollout of the objective value function so that the policy updates can be more accurate. Both schemes effectively improve the convergence speed and the attained reward. Code is available on our project page: <https://github.com/YilingQiao/diffarticulated>

Differentiable Soft Multi-body System [212]. We aim to create a realistic deformable multi-body dynamics framework, in which soft articulated robots can be simulated to learn powerful control policies. Our code is available here: https://github.com/YilingQiao/diff_fem

Differentiable Hybrid Traffic Simulation [177]. In this work, we propose to adopt both microscopic and macroscopic models to create a more general traffic simulator that is differentiable. This kind of hybrid approach has already been proven to simulate large-scale traffic environments much more efficiently than either of the approaches while maintaining integrity. <https://github.com/SonSang/diff-hybrid-traffic-sim>

Editable Geometry and Physics from Monocular RGB Videos [269]. In this chapter, we design an end-to-end differentiable rendering and simulation pipeline that can directly learn the geometry and dynamics from a monocular video. Our code is available here: <https://github.com/gaoalexander/neuphysics>

Physics Augmented Continuum Neural Radiance Fields for Geometry-Agnostic System Identification [272]. PAC-NeRF is a novel approach to estimating both the unknown geometry and physical parameters of highly dynamic objects from multi-view videos. We design PAC-NeRF to only ever produce physically plausible states by enforcing the neural radiance field to follow the conservation laws of continuum mechanics. Our code is available here: <https://github.com/gaoalexander/neuphysics>

`//github.com/xuan-li/PAC-NeRF`

Differentiable Analog Quantum Computing [202]. Differentiable analog quantum computing can be deemed as a special form of differentiable physics at the quantum scale. We develop a sampling technique to compute gradients on analog quantum machines. Our code is available here: <https://github.com/YilingQiao/diffquantum>

Dynamic Mesh-Aware Radiance Fields [379]. We integrate the conventional mesh-based graphics pipeline with the radiance field representation. We can interact with mesh and NeRF in real-time under our hybrid ray-tracing rendering and simulation pipeline. Our code is available here: <https://github.com/YilingQiao/DMRF>

Physically Consistent Perception of Hand-Object Interactions with Differentiable Priors [380]. We integrate differentiable rendering and physics priors to estimate hand-object-interaction. It can also generalize to filtering-based methods and human-pose estimation. Here is the project page: <https://handypriors.github.io/>.

10.2 Future Work

Based on my prior experience in learning, simulation, and rendering, I plan to continue contributing to AI, Robotics and Visual Computing communities. The recent success of large pre-trained models has demonstrated the potential for artificial general intelligence (AGI). However, deploying such powerful AGI in the real world without sufficient training and verification in simulation environments could be dangerous, especially in interactions with humans. With this concern in mind, I am committed to develop a simulation platform suitable for AGI development over the next five years. Ideally, this platform should be machine-learning-friendly (consistent

with my previous work), controllable (enforcing safety constraints), general (handling complex real-life tasks), and scalable (catering to large model sizes and data volumes). Simultaneously, I will explore the application of machine learning techniques to solve simulation and physics problems. For instance, leveraging differentiable dynamics can aid in structural design, robot control, and addressing broader scientific challenges, leading to more interdisciplinary projects.

Physics-Informed Artificial General Intelligence (AGI). We have witnessed very impressive progress in large-scale and multi-modal foundation/generative models recently. However, these models are usually trained on passive data such as texts and images. The generated content does not come with guarantees for physics correctness, posing a potential risk for wider use. I plan to train these models in a physically-based simulation environment and/or add explicit physics constraints to ensure their correctness. By doing so, we can improve language models' ability in physics-related reasoning. For example, we recently organized a CoRL workshop [381] on how to train a generalist robot, which garnered much attention and constructive discussion. This white paper [382] describes how to combine simulation and large language models (LLM) to create more powerful embodied AI agents. By injecting physics rules and information, we can potentially improve the safety of AI, align trained models with the physical world, and pave the way for human-robot interaction.

A Uniform and Efficient Simulation Platform for AGI. A generalist robot would require a general simulation environment. Nowadays, simulation algorithms usually only specify a certain object type. We do not have a general enough engine to cope with even our daily scenarios, such as a robot doing dishwashing using a towel. Based on my experience dealing with a wide range of dynamic systems, I want to develop a uniform and efficient simulation platform that can faithfully

simulate our real world. This requires an accurate and general contact coupling formulation for different materials (rigid, soft, fluids, etc.). Moreover, to train large models with extensive data, the simulation needs to have high throughput, ideally compatible with modern acceleration hardware (GPU, TPU, etc.). Designing simulation algorithms with massive parallelization is, therefore, an urgent task. Besides breeding AGI, such a general and easy-to-use platform can also greatly facilitate animation, VR/AR, and robotics applications.

Integrated Simulation and Learning for Scientific Exploration and Advancements. Real-world scientific experiments are usually expensive, time-consuming, and risky. Efficient simulation and learning algorithms can help accelerate progress in broader scientific problems. We already have initial examples and theories showing that differentiable quantum computing can improve optimization and control problems. Our next step is to scale it up and attempt to solve some real-world chemistry or physics problems. For example, I will continue our quantum-related projects and solve tasks in bigger condensed matter physics systems (e.g. ground-state problems). And we can also use our algorithm to build better quantum computers by designing more efficient gates.

Part V

Appendix

Appendix A: Scalable Differentiable Physics for Learning and Control

Mass Matrix

For a rigid body i with rotation $\mathbf{r} = (\phi, \theta, \psi)^\top$ and translation $\mathbf{t} = (t_x, t_y, t_z)^\top$, the generalized coordinates are $\mathbf{q} = [\mathbf{r}^\top, \mathbf{t}^\top]^\top \in \mathbb{R}^6$. Its kinetic energy can be computed by $\mathbf{E} = \frac{1}{2} \dot{\mathbf{q}}^\top \hat{\mathbf{M}} \dot{\mathbf{q}}$, where $\hat{\mathbf{M}}$ is a diagonal blocked mass matrix composed of angular and linear inertia,

$$\hat{\mathbf{M}} = \begin{bmatrix} \mathcal{I}_a & \mathbf{0} \\ \mathbf{0} & \mathcal{I}_l \end{bmatrix}. \quad (\text{A.0.0.1})$$

The linear inertia $\mathcal{I}_l$ is simply $m\mathbf{I}_{3 \times 3}$, where m is the total mass of the rigid body.

When the rigid body's distribution is approximated by a set of particles, an angular inertial is given by,

$$\mathcal{I}' = \sum_i m_i (\mathbf{p}_i^\top \mathbf{p}_i \mathbf{I}_{3 \times 3} - \mathbf{p}_i \mathbf{p}_i^\top), \quad (\text{A.0.0.2})$$

where m_i is the mass of the particle i ; $\mathbf{p}_i$ is the vector from the center of mass to this particle. Note that this angular inertia corresponds to the axis-angle velocity ω in the *world frame*. The angular momentum is $\mathcal{I}'\omega$. However, We cannot directly use this formula since we have to represent the angular velocity in terms of velocities of Euler angles.

We used the RPY convention for Euler angle representation: given the Euler angle $\mathbf{r} =$

$(\phi, \theta, \psi)^\top$, the rigid body will first rotate about the Z axis by ψ , then rotate about the new Y' axis by θ , and finally rotate about the new X'' axis by ϕ .

By the above definition, the angular velocity vector Ω can be represented by Euler angles,

$$\Omega = \dot{\psi}\mathbf{e}_Z + \dot{\theta}\mathbf{e}_{Y'} + \dot{\phi}\mathbf{e}_{X''} \quad (\text{A.0.0.3})$$

We convert the angular velocity in local frame back to the world frame so that it matches with the angular inertia in the world frame:

$$\begin{aligned} \Omega &= (\cos \theta \cos \psi \dot{\phi} - \sin \psi \dot{\theta})\mathbf{e}_X \\ &+ (\cos \theta \cos \psi \dot{\phi} + \cos \psi \dot{\theta})\mathbf{e}_Y \\ &+ (-\sin \theta \dot{\phi} + \dot{\psi})\mathbf{e}_Z \end{aligned} \quad (\text{A.0.0.4})$$

The matrix form is given as:

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \cos \theta \cos \psi & -\sin \psi & 0 \\ \cos \theta \sin \psi & \cos \psi & 0 \\ -\sin \theta & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \quad (\text{A.0.0.5})$$

We denote this transformation as $\omega = \mathbf{T}\dot{\mathbf{r}}$, so the angular momentum is reformed as $\mathcal{I}'\omega = \mathcal{I}'\mathbf{T}\dot{\mathbf{r}}$. Therefore, in the Euler angle representation, the angular inertia becomes

$$\mathcal{I}_a = \mathbf{T}^\top \mathcal{I}' \mathbf{T}. \quad (\text{A.0.0.6})$$

The new mass matrix $\hat{\mathbf{M}}$ for the generalized coordinates is,

$$\hat{\mathbf{M}} = \begin{bmatrix} \mathbf{T}^\top \mathcal{I}' \mathbf{T} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{3 \times 3} \end{bmatrix}. \quad (\text{A.0.0.7})$$

Represent a Vertex using Generalized Coordinates

For a vertex p attached to the rigid body, its coordinate in the body frame is $\mathbf{p}_0$. The origin of the body frame is set to be the center of mass (COM). $\mathbf{p}_0 = (p_x, p_y, p_z)^\top$ is the relative displacement of p w.r.t. the COM in the first frame of simulation. The coordinates of the vertex in the world frame are then given by,

$$\mathbf{p} = \mathbf{f}(\mathbf{q}) = [\mathbf{r}] \mathbf{p}_0 + \mathbf{t}, \quad (\text{A.0.0.8})$$

where $[\mathbf{r}]$ is a rotation matrix represented by the Euler angle $\mathbf{r} = (\phi, \theta, \psi)^\top$. The corresponding rotation matrix $[\mathbf{r}] = \mathbf{R}_{3 \times 3}$ is computed by

$$\mathbf{R}_{11} = \cos \theta \cos \psi$$

$$\mathbf{R}_{12} = -\cos \phi \sin \psi + \sin \phi \sin \theta \cos \psi$$

$$\mathbf{R}_{13} = \sin \phi \sin \psi + \cos \phi \sin \theta \cos \psi$$

$$\mathbf{R}_{21} = \cos \theta \sin \psi$$

$$\mathbf{R}_{22} = \cos \phi \cos \psi + \sin \phi \sin \theta \sin \psi$$

$$\mathbf{R}_{23} = -\sin \phi \cos \psi + \cos \phi \sin \theta \sin \psi$$

$$\mathbf{R}_{31} = -\sin \theta$$

$$\mathbf{R}_{32} = \sin \phi \cos \theta$$

$$\mathbf{R}_{33} = \cos \phi \cos \theta$$

Computation of Derivatives

To backpropagate the gradients at vertex $\mathbf{p}$ to the generalized coordinates $\mathbf{q}$, we need to compute the derivatives $\partial \mathbf{f}(\mathbf{q})/\partial \mathbf{q}$.

The coordinates of a vertex $(x, y, z)^\top = \mathbf{f}([\phi, \theta, \psi, t_x, t_y, t_z]^\top)$ are given by Equation [A.0.0.8](#).

Therefore, the Jacobian matrix for the partial derivatives is:

$$\nabla \mathbf{f} = \begin{bmatrix} \frac{\partial x}{\partial \phi} & \frac{\partial x}{\partial \theta} & \frac{\partial x}{\partial \psi} & 1 & 0 & 0 \\ \frac{\partial y}{\partial \phi} & \frac{\partial y}{\partial \theta} & \frac{\partial y}{\partial \psi} & 0 & 1 & 0 \\ \frac{\partial z}{\partial \phi} & \frac{\partial z}{\partial \theta} & \frac{\partial z}{\partial \psi} & 0 & 0 & 1 \end{bmatrix}. \quad (\text{A.0.0.9})$$

Assuming that the relative displacement of p w.r.t. the COM in the first frame is $\mathbf{p}_0 = (p_x, p_y, p_z)$, the corresponding elements in the Jacobian matrix are:

$$\frac{\partial x}{\partial \psi} = -p_x \cos \theta \sin \psi + p_y (-\cos \phi \cos \psi - \sin \phi \sin \theta \sin \psi) + p_z (\sin \phi \cos \psi - \cos \phi \sin \theta \sin \psi)$$

$$\frac{\partial x}{\partial \theta} = -p_x \sin \theta \cos \psi + p_y \sin \phi \cos \theta \cos \psi + p_z \cos \phi \cos \theta \cos \psi$$

$$\frac{\partial x}{\partial \phi} = p_y \sin \phi \sin \psi + \cos \phi \sin \theta \cos \psi + p_z (\cos \phi \sin \psi - \sin \phi \sin \theta \cos \psi)$$

$$\frac{\partial y}{\partial \psi} = p_x \cos \theta \cos \psi + p_y (-\cos \phi \sin \psi + \sin \phi \sin \theta \cos \psi) + p_z (\sin \phi \sin \psi + \cos \phi \sin \theta \cos \psi)$$

$$\frac{\partial y}{\partial \theta} = -p_x \sin \theta \sin \psi + p_y \sin \phi \cos \theta \sin \psi + p_z \cos \phi \cos \theta \sin \psi$$

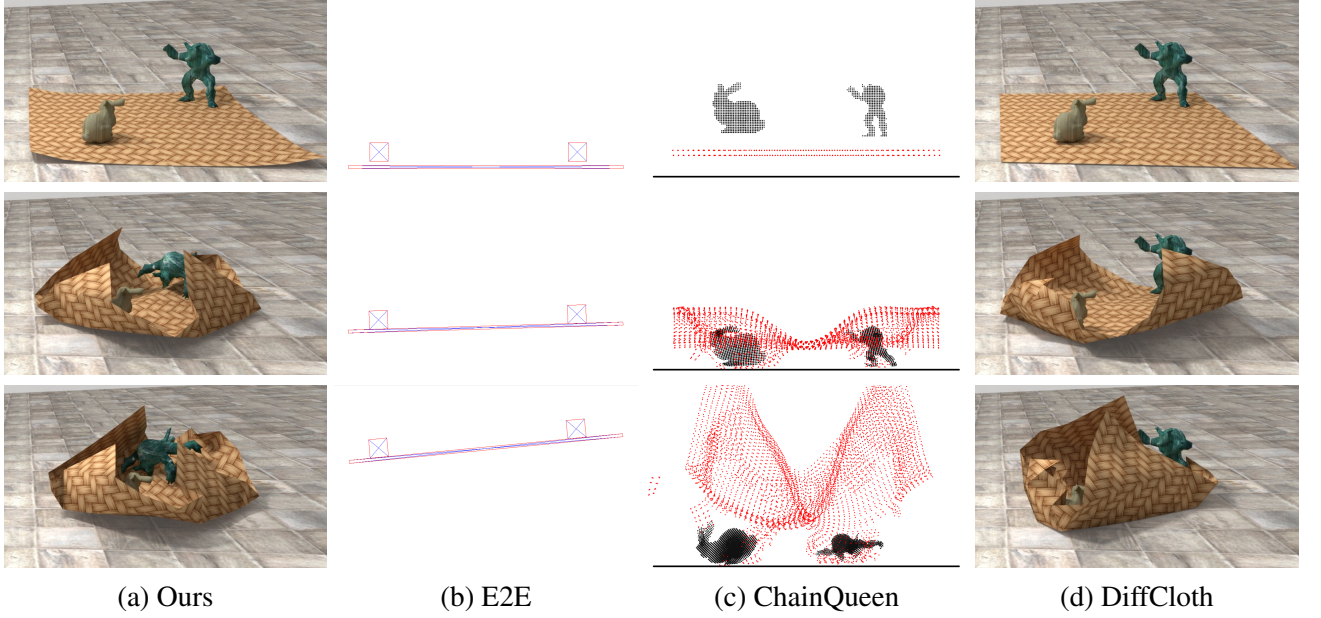


Figure A.1: Qualitative comparison on complex models and interactions. In this scene, we simulate two complex rigid bodies, bunny and armadillo on a cloth. (a) is the simulation result of our method, and the comparisons, (b) is E2E [2]; (c) is ChainQueen [6]; (d) is DiffCloth [3]. Our method gets the most reasonable results because we can deal with both rigid body and cloth dynamics. Meanwhile, the mesh representation enables us to model different objects precisely.

$$\frac{\partial y}{\partial \phi} = p_y(-\sin \phi \cos \psi + \cos \phi \sin \theta \sin \psi) + p_z(-\cos \phi \cos \psi - \sin \phi \sin \theta \sin \psi)$$

$$\frac{\partial z}{\partial \psi} = 0$$

$$\frac{\partial z}{\partial \theta} = -p_x \cos \theta - p_y \sin \phi \sin \theta - p_z \cos \phi \sin \theta$$

$$\frac{\partial z}{\partial \phi} = p_y \cos \phi \cos \theta - p_z \sin \phi \cos \theta$$

Qualitative Comparisons

Figure A.1 shows qualitative comparisons of two complex rigid bodies interacting with a soft cloth.

In this comparison, our differentiable physics gets the most realistic results, that both the bunny and armadillo are lifted from the table. E2E cannot simulate the detailed motion of the objects because it only supports basic primitives. ChainQueen cannot preserve the *rigid*

property of bunny and armadillo because these objects are composed of small particles during the simulation and obey deformable body dynamics. Moreover, the cloth would break up because the MPM method cannot keep connectivity information faithfully during the simulation. Diffcloth is able to model the motion of cloth, but the cloth cannot have an impact on the bunny and armadillo, therefore they can not be lifted.

Appendix B: Efficient Differentiable Simulation of Articulated Bodies

Discussion of RL Applications

For the policy enhancement method used in n -link pendulum, we use off-policy training and weigh the relevance of each sample according to the distance to the current action:

$$w = \exp(-\|\mathbf{a} - \mathbf{a}_0\|) \tag{B.0.0.1}$$

where $\mathbf{a}$ is the policy network output, and $\mathbf{a}_0$ is the action sampled from the history pool. For every epoch, we sample from the environment 1,000 times and train the model after each sample. We train both models from the simplest 1-link to the most difficult 7-link setting, each experiment repeated 10 times to obtain stable statistics. In Table B.1, we report the statistics of the maximum return for pendulums of different numbers of links. As mentioned in the main text, the performance of MBPO degrades while ours does not. The reason behind it is that, during the policy update, our method does not need to fully depend on the critic function. By making use of the gradients as shown in Eq. 3.5.0.3, the policy network can first optimize the short-term reward ($\partial r / \partial a$) before the critic is up to date. When the critic network is able to reveal the true value of the state-action pair, the policy can then converge as well to the long term goal. On the other hand, traditional RL methods heavily depend on the convergence of the critic network, thus having a hard time

searching for the best action in a large space, especially when the links are attached sequentially.

In Fig. 3.5 that shows our MuJoCo Ant experiment results, we observe an even faster convergence than MBPO, while MBPO already outperforms most of the RL algorithms. This is because by generating nearby samples, the feedback from the environment is no longer a single point but a patch as an approximation of the shape of the state transitioning function. It can greatly help the critic function learn the correct estimation of the values not only for the observation point but also nearby ones, thereby achieving faster convergence on the value estimation and eventually the policy model.

Limitations and future directions. We also observe limitations on these two proposed techniques. We observed that the sample enhancement does not provide better overall rewards in the ‘pendulum’ test, and the policy enhancement does not provide faster convergence in the ‘ant’ test. We attribute this to a number of possible causes. Sample enhancement enriches the interaction history with the environment, but it is inherently limited because extra samples can only be generated near the true sample, it may not help the agent jump out of the local minima. Policy enhancement relies heavily on the assumption that the actions drawn from the training batch are the same as or similar to the action that the policy takes at the current moment: it is considered as an on-policy learning algorithm. Therefore, it faces the same limitation, which is poor sample efficiency, as other on-policy algorithms as well. These two techniques cannot be combined together because the generated samples from the sample enhancement do not have accurate first-order gradients at their location, as mentioned in the paper.

While we have shown the two techniques to be a proof-of-concept demonstration that differentiable physics can help improve performance of reinforcement learning using an appropriate method depending on different scenarios, there remain several directions to further explore.

We hope that our preliminary investigation on differentiable physics integrated with RL, as presented in this chapter, will stimulate more study on the coupling of differentiable physics with reinforcement learning.

# of links	1	2	3	4	5	6	7
Ours	0.49± 0.01	1.93± 0.08	4.28± 0.25	7.81± 0.19	12.31± 0.16	17.78± 0.24	24.34± 0.11
MBPO	0.49± 0.01	1.90± 0.02	4.28± 0.22	7.50± 0.57	11.13± 0.99	15.02± 1.45	19.97± 2.33
Maximum	0.5	2	4.5	8	12.5	18	24.5

Table B.1: Maximum reward on n -link pendulum. Our method has higher reward scores than MBPO, as the system complexity increases.

Ablation Study for Checkpointing Scheme

In this section, we conduct an ablation study to show why we choose to save the checkpoints each time step, instead of having a lower frequency for checkpointing. We vary the intervals between checkpoints and profile the peak memory usage and backpropagation time per step. Our default design is interval=1 such that checkpoints are stored every step.

If interval= k , our method needs to compute the gradients $\overline{x_i}, \overline{x_{i+1}}, \dots, \overline{x_{i+k-1}}$ given the gradients $\overline{x_{i+k}}$ and the checkpoint x_i . The intermediate variables in step $i+k-1, i+k-2, \dots, i$ have to be resumed from x_i , taking $k, i-1, \dots, 1$ steps of forward simulations, respectively. Therefore, the total forward steps used to resume variables here are $(k+1)k/2$. If there are n simulation steps in total, the average time of backpropagation per step should be

$$(k+1)/2 \cdot t_{forward} + t_{backward}, \quad (\text{B.0.0.2})$$

where $t_{simulation}, t_{adjoint}$ are the time for one step of simulation and adjoint method, respectively.

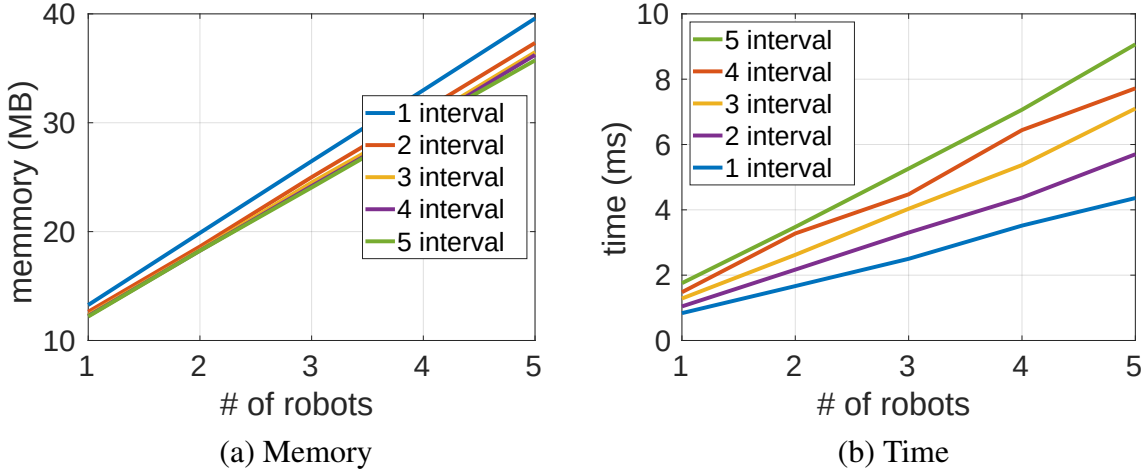


Figure B.1: Ablation study for the checkpointing scheme.

On the other hand, the peak memory usage is

$$n/k \cdot m_{checkpoint} + m_{simulation}, \quad (\text{B.0.0.3})$$

where $m_{checkpoint}$ is the size of one checkpoint; $m_{simulation}$ is the memory usage of one step, which is the size of all intermediate variables. Since $m_{simulation} \gg m_{checkpoint}$, the increase of k would not significantly reduce the memory usage but will slow down the backpropagation linearly.

In Figure B.1, the experiments have similar setup as in Section 3.6.1 where we simulate 10 Laikago robots for 5,000 steps. The memory and the time scale linearly w.r.t. the number of the robots. When the checkpoint intervals get larger, the memory usage decreases gradually to the lower bound of $m_{simulation}$, but the average backpropagation time increases linearly. Since the memory usage is low enough even for interval $k = 1$, we choose to checkpoint every step to have a faster speed and set the interval $k = 1$.

The Adjoints of Contact Solver

In the collision solver, we construct a Mixed Linear Complementarity Problem (MLCP) [140]:

$$\begin{aligned} \mathbf{a} &= \mathbf{A}\mathbf{x} + \mathbf{b} \\ \text{s.t. } \mathbf{0} &\leq \mathbf{a} \perp \mathbf{x} \leq \mathbf{0} \text{ and } \mathbf{c}_- \leq \mathbf{x} \leq \mathbf{c}_+, \end{aligned} \tag{B.0.0.4}$$

where $\mathbf{x}$ is the new collision-free state, $\mathbf{A}$ is the inertial matrix, $\mathbf{b}$ contains the relative velocities, and $\mathbf{c}_-, \mathbf{c}_+$ are the lower bound and upper bound constraints, respectively. In the forward pass, $\mathbf{A}, \mathbf{b}$ are known variables, and the projected Gauss-Seidel (PGS) method is used to solve for $\mathbf{x}$.

In Algorithm 4, we are computing the adjoint of $\mathbf{A}, \mathbf{b}$ given the adjoint $\bar{\mathbf{x}}$. For simplicity, we assume there are no constraints $\mathbf{c}_-, \mathbf{c}_+$ and it is now a pure Gauss-Seidel method which solves $\mathbf{A}\mathbf{x} + \mathbf{b} = \mathbf{0}$. Basically, we store variables in $\mathbf{V}_d, \mathbf{V}_{x1}$, and $\mathbf{V}_{x2}$. In the backward pass, we load the variables and compute the adjoints in a reverse order.

Model Mismatch/Transplant

Our articulated body simulation has similar functionality to MuJoCo [122] in terms of physical and contact modeling. We first trained the ‘MuJoCo Ant’ model using our simulation as the environment. Upon convergence, we directly tested our model in MuJoCo, while keeping all robot configuration the same. As shown in Table B.2, without any retraining, the trained model is able to obtain non-trivial reward in MuJoCo.

Algorithm 4 Adjoint of Gaussian-Siedel Solver

Input: matrix $\mathbf{A}$, vector $\mathbf{b}$, adjoint $\bar{\mathbf{x}}$, iterations $niter$
Output: adjoints $\bar{\mathbf{A}}, \bar{\mathbf{b}}$
Initialize vectors $\mathbf{V}_d(niter, rows(\mathbf{A}))$,
 $\mathbf{V}_{x1}(niter, rows(\mathbf{A}))$, $\mathbf{V}_{x2}(niter, rows(\mathbf{A}))$.
forward pass
for $t = 1$ **to** $niter$ **do**
 $\mathbf{V}_{x1}(t) = \mathbf{x}$
 for $i = 1$ **to** $rows(\mathbf{A})$ **do**
 $d = 0$
 for $j = 1$ **to** $rows(\mathbf{A})$ **do**
 if $j \neq i$ **then**
 $d = d + \mathbf{A}(i, j) * \mathbf{x}(j)$
 $\mathbf{V}_d(t, i) = d$
 $\mathbf{x}(i) = (\mathbf{b}(i) - d) / \mathbf{A}(i, i)$
 $\mathbf{V}_{x2}(t, i) = \mathbf{x}(i)$
start backward process
 $\bar{\mathbf{A}} = \mathbf{0}, \bar{\mathbf{b}} = \mathbf{0}$
for $t=1$ **to** $niter$ **do**
 for $i = rows(\mathbf{A})$ **to** 1 **do**
 $\bar{d} = 0$
 $\bar{d} = \bar{d} - \bar{\mathbf{x}}(i) / \mathbf{A}(i, i)$
 $\bar{\mathbf{b}} = \bar{\mathbf{b}} + \bar{\mathbf{x}}(i) / \mathbf{A}(i, i)$
 $\bar{\mathbf{A}}(i, i) = \bar{\mathbf{A}}(i, i) - \bar{\mathbf{x}}(i) * [\mathbf{b}(i) - \mathbf{V}_d(t, i)] / \mathbf{A}(i, i)^2$
 $\bar{\mathbf{x}}(i) = 0$
 for $j = rows(\mathbf{A})$ **to** $i+1$ **do**
 $\bar{\mathbf{A}}(i, j) = \bar{\mathbf{A}}(i, j) d + \bar{d} * \mathbf{V}_{x1}(t, j)$
 $\bar{\mathbf{x}}(j) = \bar{\mathbf{x}}(j) + \bar{d} * \mathbf{A}(i, j)$
 for $j = i - 1$ **to** 1 **do**
 $\bar{\mathbf{A}}(i, j) = \bar{\mathbf{A}}(i, j) d + \bar{d} * \mathbf{V}_{x2}(t, j)$
 $\bar{\mathbf{x}}(j) = \bar{\mathbf{x}}(j) + \bar{d} * \mathbf{A}(i, j)$

Method	Reward in MuJoCo
w/o pretraining	868 $\pm$ 77
w/ pretraining (Ours)	1735$\pm$1066

Table B.2: Model transplant results on MuJoCo Ant. After pretraining in our simulation environment, the model can be directly transplanted to a new environment, MuJoCo, with non-trivial reward. This shows that our simulator behaves similarly with MuJoCo.

Other Limitations and Future Directions

In addition to some possible directions to further explore, as mentioned in Appendix B, our method can benefit from additional enhancement. The current simulator can potentially

provide more options for users. For example, we have not provided options to select different contact models. The current LCP formula using iterative solver is fast but sometimes not accurate enough. Implementing extra contact models like exact LCP solvers or convex soft contact model can help the simulator adapt to more applications. Moreover, we assume that each link is a rigid body. It would be natural to extend this work to support deformable objects such as cloth [4, 122].

We also plan to build on top of the current simulator to provide more comprehensive tool sets for users. Furthermore, it is worth exploring how differentiable physics can improve the sampling strategies and optimization of deep reinforcement learning models. Lastly, we hope to deploy our algorithms on real-world robots to perform a wide range of complex tasks.

Code for Profiling Autodiff Tools

ADF. We post below the code to run the backpropagation of ADF [142]. As we described in Section 3.6.1, ADF can be used to run the forward simulation of articulated body dynamics but fails to complete the backpropagation in a reasonable time. We found out that ADF will greatly slow down when the computation graph goes deeper. Here is an example where we repeat a simple operation iteratively. When the iteration number is 28, it already takes a long time to backpropagate. In articulated body simulations, the depth of computation is far larger than 28, so ADF fails to compute the corresponding gradients.

```
#include <autodiff/reverse.hpp>

using namespace autodiff;

// The single-variable function for
```

```

// which derivatives are needed

var f(var x)
{
    int num_iter = 28;

    for (int i=0; i<num_iter; i++)

        x = x + 1 / x;

    return x;
}

int main()
{
    var x = 2.0;    // the input variable x
    var u = f(x);   // the output variable u

    // evaluate the derivative of u
    // with respect to x

    auto [ux] = derivatives(u, wrt(x));

}

```

JAX. JAX is based on Python and is hard to integrate into our C++ simulator. Therefore, we write simplified Python code to do the profiling. The input parameters to this code are the number of robots and simulation length. Each robot in the simulation environment of Section [3.6.1](#) has 37

DoFs, so we multiply 37 by the number of robots and get the dimension of the states variable.

The operations in one simulation loop are far simpler than the articulated body dynamics.

```
from jax import grad

import jax.numpy as jnp

from jax import jit

import numpy as np

import sys


class RunBW(object):

    def __init__(self, dim, n_step):

        self.dim = dim

        self.n_step = n_step


    def test(self, x):

        dim = self.dim

        n_step = self.n_step

        for s in range(n_step):

            var = []

            for k in range(dim):

                var.append(x[k])

            for i in range(dim):

                k = i % (dim - 1) + 1
```

```

        var[k] = var[k-1] + 1/var[k]

    var = [jnp.expand_dims(v, 0) for
           v in var]

    x = jnp.concatenate(var, 0)

    return jnp.sum(x)

if __name__ == '__main__':
    dim = 37 * int(sys.argv[1])
    n_step = int(sys.argv[2])
    rb = RunBW(dim, n_step)
    ini_state = np.ones([dim])
    grad_test = grad(rb.test)
    result = grad_test(ini_state)
    result = rb.test(ini_state)

```

Differentiation of Articulated Body Algorithm

This section describes the list of adjoint operations. We start with basic operators and go over all adjoints used in our implementation. $\overline{(\cdot)}$ is denoted as the adjoint of a variable.

Scalar multiply:

$$\begin{aligned} a &= bc \\ \frac{\partial \phi}{\partial b} &= \frac{\partial \phi}{\partial a} \frac{\partial a}{\partial b} \\ \bar{b} &= \bar{a}c, \bar{c} = b\bar{a} \end{aligned} \tag{B.0.0.5}$$

Matrix vector multiply:

$$\begin{aligned} \mathbf{v} &= \mathbf{m}\mathbf{a} \\ \bar{\mathbf{m}}_{ij} &= \bar{\mathbf{v}}_i \mathbf{a}_j, \bar{\mathbf{a}}_j = \sum \bar{\mathbf{v}}_k \mathbf{m}_{kj} \\ \bar{\mathbf{m}} &= \bar{\mathbf{v}}\mathbf{a}^T \\ \bar{\mathbf{a}} &= \mathbf{m}^T \bar{\mathbf{v}} \end{aligned} \tag{B.0.0.6}$$

Matrix multiply:

$$\begin{aligned} \mathbf{c} &= \mathbf{a}\mathbf{b} \\ \bar{\mathbf{a}} &= \bar{\mathbf{c}}\mathbf{b}^T \\ \bar{\mathbf{b}} &= \mathbf{a}^T \bar{\mathbf{c}} \end{aligned} \tag{B.0.0.7}$$

Matrix $\mathbf{A}^T \mathbf{B} \mathbf{A}$:

$$\mathbf{c} = \mathbf{a}^T \mathbf{b} \mathbf{a}$$

$$\bar{\mathbf{a}} = \mathbf{b}^T \mathbf{a} \bar{\mathbf{c}} + \mathbf{b} \mathbf{a} \bar{\mathbf{c}}^T \quad (\text{B.0.0.8})$$

$$\bar{\mathbf{b}} = \mathbf{a} \bar{\mathbf{c}} \mathbf{a}^T$$

Matrix inverse:

$$\mathbf{b} = \mathbf{a}^{-1}$$

$$\frac{\partial \mathbf{b}}{\partial \mathbf{q}} = -\mathbf{b} \frac{\partial \mathbf{a}}{\partial \mathbf{q}} \mathbf{b} = -\sum_k \sum_t \mathbf{b}_{ij} \frac{\partial \mathbf{a}_{jk}}{\partial \mathbf{q}} \mathbf{b}_{kt} \quad (\text{B.0.0.9})$$

$$\bar{\mathbf{a}}_{jk} = -\sum_i \sum_t \mathbf{b}_{ij} \mathbf{b}_{kt} \bar{\mathbf{b}}_{it}$$

$$\bar{\mathbf{a}} = -\mathbf{b}^T \bar{\mathbf{b}} \mathbf{b}^T$$

Multiply three matrices:

$$\mathbf{m} = \mathbf{a} \mathbf{b} \mathbf{c}$$

$$\bar{\mathbf{a}} = \bar{\mathbf{m}} \mathbf{c}^T \mathbf{b}^T$$

(B.0.0.10)

$$\bar{\mathbf{b}} = \mathbf{a}^T \bar{\mathbf{m}} \mathbf{c}^T$$

$$\bar{\mathbf{c}} = \mathbf{b}^T \mathbf{a}^T \bar{\mathbf{m}}$$

Vector cross product:

$$\mathbf{v} = \mathbf{a} \times \mathbf{b} = [\mathbf{a}_2\mathbf{b}_3 - \mathbf{a}_3\mathbf{b}_2, \mathbf{a}_3\mathbf{b}_1 - \mathbf{a}_1\mathbf{b}_3, \mathbf{a}_1\mathbf{b}_2 - \mathbf{a}_2\mathbf{b}_1]$$

$$\bar{\mathbf{a}} = [-\bar{\mathbf{v}}_2\mathbf{b}_3 + \bar{\mathbf{v}}_3\mathbf{b}_2, \bar{\mathbf{v}}_1\mathbf{b}_3 - \bar{\mathbf{v}}_3\mathbf{b}_1, -\bar{\mathbf{v}}_1\mathbf{b}_2 + \bar{\mathbf{v}}_2\mathbf{b}_1] = -\bar{\mathbf{v}} \times \mathbf{b} \quad (\text{B.0.0.11})$$

$$\bar{\mathbf{b}} = [\bar{\mathbf{v}}_2\mathbf{a}_3 - \bar{\mathbf{v}}_3\mathbf{a}_2, -\bar{\mathbf{v}}_1\mathbf{a}_3 + \bar{\mathbf{v}}_3\mathbf{a}_1, \bar{\mathbf{v}}_1\mathbf{a}_2 - \bar{\mathbf{v}}_2\mathbf{a}_1] = \bar{\mathbf{v}} \times \mathbf{a}$$

Vector norm:

$$a = \sqrt{\mathbf{b} \cdot \mathbf{b}}$$

$$\frac{\partial a}{\partial \mathbf{q}} = -\frac{1}{2} \frac{1}{\sqrt{\mathbf{b} \cdot \mathbf{b}}} \left(\frac{\partial \mathbf{b}}{\partial \mathbf{q}} \cdot \mathbf{b} + \mathbf{b} \cdot \frac{\partial \mathbf{b}}{\partial \mathbf{q}} \right) = \frac{1}{a} \frac{\partial \mathbf{b}}{\partial \mathbf{q}} \cdot \mathbf{b} \quad (\text{B.0.0.12})$$

$$\bar{\mathbf{b}} = \frac{\bar{a}}{a} \mathbf{b}$$

Spatial transform: apply():

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = st_{apply} \left(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}, \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{E}\mathbf{a}_1 \\ \mathbf{E}(\mathbf{a}_2 - \mathbf{r} \times \mathbf{a}_1) \end{bmatrix} \quad (\text{B.0.0.13})$$

$$\begin{bmatrix} \bar{\mathbf{E}} \\ \bar{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \bar{\mathbf{b}}_1^T + (\mathbf{a}_2 - \mathbf{r} \times \mathbf{a}_1) \bar{\mathbf{b}}_2^T \\ \mathbf{E}^T \bar{\mathbf{b}}_2 \times \mathbf{a}_1 \end{bmatrix}, \begin{bmatrix} \bar{\mathbf{a}}_1 \\ \bar{\mathbf{a}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E}^T \bar{\mathbf{b}}_1 - \mathbf{E}^T \bar{\mathbf{b}}_2 \times \mathbf{r} \\ \mathbf{E}^T \bar{\mathbf{b}}_2 \end{bmatrix} \quad (\text{B.0.0.14})$$

Spatial transform: `apply-inv()`:

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = st_{apply-inv} \left(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}, \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{E}^T \mathbf{a}_1 \\ \mathbf{E}^T \mathbf{a}_2 + \mathbf{r} \times (\mathbf{E}^T \mathbf{a}_1) \end{bmatrix} \quad (\text{B.0.0.15})$$

$$\begin{bmatrix} \bar{\mathbf{E}} \\ \bar{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 (\bar{\mathbf{b}}_1 + \bar{\mathbf{b}}_2 \times \mathbf{r})^T + \mathbf{a}_2 \bar{\mathbf{b}}_2^T \\ -\bar{\mathbf{b}}_2 \times \mathbf{E}^T \mathbf{a}_1 \end{bmatrix}, \begin{bmatrix} \bar{\mathbf{a}}_1 \\ \bar{\mathbf{a}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E} (\bar{\mathbf{b}}_1 + \bar{\mathbf{b}}_2 \times \mathbf{r}) \\ \mathbf{E} \bar{\mathbf{b}}_2 \end{bmatrix} \quad (\text{B.0.0.16})$$

Spatial transform: `apply-trans()`:

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = st_{apply-trans} \left(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}, \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{E}^T \mathbf{a}_1 + \mathbf{r} \times (\mathbf{E}^T \mathbf{a}_2) \\ \mathbf{E}^T \mathbf{a}_2 \end{bmatrix} \quad (\text{B.0.0.17})$$

$$\begin{bmatrix} \bar{\mathbf{E}} \\ \bar{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{b}}_1 \mathbf{a}_1^T + (\bar{\mathbf{b}}_1 \times \mathbf{r} + \bar{\mathbf{b}}_2) \mathbf{a}_2^T \\ -\bar{\mathbf{b}}_1 \times \mathbf{E}^T \mathbf{a}_2 \end{bmatrix}, \begin{bmatrix} \bar{\mathbf{a}}_1 \\ \bar{\mathbf{a}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E} \bar{\mathbf{b}}_1 \\ \mathbf{E} (\bar{\mathbf{b}}_1 \times \mathbf{r} + \bar{\mathbf{b}}_2) \end{bmatrix} \quad (\text{B.0.0.18})$$

Spatial transform: `apply-invtrans()`:

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = st_{apply-invtrans} \left(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}, \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{E}^T (\mathbf{a}_1 - \mathbf{r} \times \mathbf{a}_2) \\ \mathbf{E}^T \mathbf{a}_2 \end{bmatrix} \quad (\text{B.0.0.19})$$

$$\begin{bmatrix} \bar{\mathbf{E}} \\ \bar{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} (\mathbf{a}_1 - \mathbf{r} \times \mathbf{a}_2) \bar{\mathbf{b}}_1^T + \mathbf{a}_2 \bar{\mathbf{b}}_2^T \\ \mathbf{E} \bar{\mathbf{b}}_1 \times \mathbf{a}_2 \end{bmatrix}, \begin{bmatrix} \bar{\mathbf{a}}_1 \\ \bar{\mathbf{a}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E} \bar{\mathbf{b}}_1 \\ -\mathbf{E} \bar{\mathbf{b}}_1 \times \mathbf{r} + \mathbf{E} \bar{\mathbf{b}}_2 \end{bmatrix} \quad (\text{B.0.0.20})$$

Spatial transform: `multiply()`:

$$\begin{bmatrix} \mathbf{E}_0 \\ \mathbf{r}_0 \end{bmatrix} = st_{multiply} \left(\begin{bmatrix} \mathbf{E}_1 \\ \mathbf{r}_1 \end{bmatrix}, \begin{bmatrix} \mathbf{E}_2 \\ \mathbf{r}_2 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{E}_1 \mathbf{E}_2 \\ \mathbf{r}_2 + \mathbf{E}_2^T \mathbf{r}_1 \end{bmatrix} \quad (\text{B.0.0.21})$$

$$\begin{bmatrix} \bar{\mathbf{E}}_1 \\ \bar{\mathbf{r}}_1 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{E}}_0 \mathbf{E}_2^T \\ \mathbf{E}_2 \bar{\mathbf{r}}_0 \end{bmatrix}, \begin{bmatrix} \bar{\mathbf{E}}_2 \\ \bar{\mathbf{r}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E}_1^T \bar{\mathbf{E}}_0 + \mathbf{r}_1 \bar{\mathbf{r}}_0^T \\ \bar{\mathbf{r}}_0 \end{bmatrix} \quad (\text{B.0.0.22})$$

Spatial motion: `crossm()`:

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{v}_0 \end{bmatrix} = sm_{crossm} \left(\begin{bmatrix} \mathbf{w}_1 \\ \mathbf{v}_1 \end{bmatrix}, \begin{bmatrix} \mathbf{w}_2 \\ \mathbf{v}_2 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{w}_1 \times \mathbf{w}_2 \\ \mathbf{w}_1 \times \mathbf{v}_2 + \mathbf{v}_1 \times \mathbf{w}_2 \end{bmatrix} \quad (\text{B.0.0.23})$$

$$\begin{bmatrix} \bar{\mathbf{w}}_1 \\ \bar{\mathbf{v}}_1 \end{bmatrix} = \begin{bmatrix} -\bar{\mathbf{w}}_0 \times \mathbf{w}_2 - \bar{\mathbf{v}}_0 \times \mathbf{v}_2 \\ -\bar{\mathbf{v}}_0 \times \mathbf{w}_2 \end{bmatrix}, \quad \begin{bmatrix} \bar{\mathbf{w}}_2 \\ \bar{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{w}}_0 \times \mathbf{w}_1 + \bar{\mathbf{v}}_0 \times \mathbf{v}_1 \\ \bar{\mathbf{v}}_0 \times \mathbf{w}_1 \end{bmatrix} \quad (\text{B.0.0.24})$$

Spatial motion: crossf():

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{v}_0 \end{bmatrix} = sm_{crossf}\left(\begin{bmatrix} \mathbf{w}_1 \\ \mathbf{v}_1 \end{bmatrix}, \begin{bmatrix} \mathbf{w}_2 \\ \mathbf{v}_2 \end{bmatrix}\right) = \begin{bmatrix} \mathbf{w}_1 \times \mathbf{w}_2 + \mathbf{v}_1 \times \mathbf{v}_2 \\ \mathbf{w}_1 \times \mathbf{v}_2 \end{bmatrix} \quad (\text{B.0.0.25})$$

$$\begin{bmatrix} \bar{\mathbf{w}}_1 \\ \bar{\mathbf{v}}_1 \end{bmatrix} = \begin{bmatrix} -\bar{\mathbf{w}}_0 \times \mathbf{w}_2 - \bar{\mathbf{v}}_0 \times \mathbf{v}_2 \\ -\bar{\mathbf{w}}_0 \times \mathbf{v}_2 \end{bmatrix}, \quad \begin{bmatrix} \bar{\mathbf{w}}_2 \\ \bar{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{w}}_0 \times \mathbf{w}_1 \\ \bar{\mathbf{w}}_0 \times \mathbf{v}_1 + \bar{\mathbf{v}}_0 \times \mathbf{w}_1 \end{bmatrix} \quad (\text{B.0.0.26})$$

Spatial dyad: mul-ori():

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{v}_0 \end{bmatrix} = sd_{mul-inv}\left(\begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix}, \begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix}\right) = \begin{bmatrix} \mathbf{m}_{11}\mathbf{w} + \mathbf{m}_{12}\mathbf{v} \\ \mathbf{m}_{21}\mathbf{w} + \mathbf{m}_{22}\mathbf{v} \end{bmatrix} \quad (\text{B.0.0.27})$$

$$\begin{bmatrix} \bar{\mathbf{m}}_{11} & \bar{\mathbf{m}}_{12} \\ \bar{\mathbf{m}}_{21} & \bar{\mathbf{m}}_{22} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{w}}_0 \mathbf{w}^T & \bar{\mathbf{w}}_0 \mathbf{v}^T \\ \bar{\mathbf{v}}_0 \mathbf{w}^T & \bar{\mathbf{v}}_0 \mathbf{v}^T \end{bmatrix}, \quad \begin{bmatrix} \bar{\mathbf{w}} \\ \bar{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{m}_{11}^T \bar{\mathbf{w}}_0 + \mathbf{m}_{21}^T \bar{\mathbf{v}}_0 \\ \mathbf{m}_{12}^T \bar{\mathbf{w}}_0 + \mathbf{m}_{22}^T \bar{\mathbf{v}}_0 \end{bmatrix} \quad (\text{B.0.0.28})$$

Spatial dyad: $\text{mul-inv}()$:

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{v}_0 \end{bmatrix} = sd_{mul-inv} \left(\begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix}, \begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix} \right) = \begin{bmatrix} \mathbf{m}_{11}^T \mathbf{w} + \mathbf{m}_{12}^T \mathbf{v} \\ \mathbf{m}_{21}^T \mathbf{w} + \mathbf{m}_{22}^T \mathbf{v} \end{bmatrix} \quad (\text{B.0.0.29})$$

$$\begin{bmatrix} \bar{\mathbf{m}}_{11} & \bar{\mathbf{m}}_{12} \\ \bar{\mathbf{m}}_{21} & \bar{\mathbf{m}}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{w} \bar{\mathbf{w}}_0^T & \mathbf{v} \bar{\mathbf{w}}_0^T \\ \mathbf{w} \bar{\mathbf{v}}_0^T & \mathbf{v} \bar{\mathbf{v}}_0^T \end{bmatrix}, \quad \begin{bmatrix} \bar{\mathbf{w}} \\ \bar{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{m}_{11} \bar{\mathbf{w}}_0 + \mathbf{m}_{21} \bar{\mathbf{v}}_0 \\ \mathbf{m}_{12} \bar{\mathbf{w}}_0 + \mathbf{m}_{22} \bar{\mathbf{v}}_0 \end{bmatrix} \quad (\text{B.0.0.30})$$

Spatial dyad: $\mathbf{v}\mathbf{v}^T()$:

$$\begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix} = sd_{vvT} \left(\begin{bmatrix} \mathbf{w}_1 \\ \mathbf{v}_1 \end{bmatrix}, \begin{bmatrix} \mathbf{w}_2 \\ \mathbf{v}_2 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{w}_1 \mathbf{w}_2^T & \mathbf{w}_1 \mathbf{v}_2^T \\ \mathbf{v}_1 \mathbf{w}_2^T & \mathbf{v}_1 \mathbf{v}_2^T \end{bmatrix} \quad (\text{B.0.0.31})$$

$$\begin{bmatrix} \bar{\mathbf{w}}_1 \\ \bar{\mathbf{v}}_1 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{m}}_{11}\mathbf{w}_2 + \bar{\mathbf{m}}_{12}\mathbf{v}_2 \\ \bar{\mathbf{m}}_{21}\mathbf{w}_2 + \bar{\mathbf{m}}_{22}\mathbf{v}_2 \end{bmatrix}, \quad \begin{bmatrix} \bar{\mathbf{w}}_2 \\ \bar{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{m}}_{11}^T\mathbf{w}_1 + \bar{\mathbf{m}}_{21}^T\mathbf{v}_1 \\ \bar{\mathbf{m}}_{21}^T\mathbf{w}_1 + \bar{\mathbf{m}}_{22}^T\mathbf{v}_1 \end{bmatrix} \quad (\text{B.0.0.32})$$

Spatial dyad: $\mathbf{v}_\times$:

$$\begin{bmatrix} \mathbf{m}_{00} & \mathbf{m}_{01} & \mathbf{m}_{02} \\ \mathbf{m}_{10} & \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{20} & \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix} = ([\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]^T)_\times = \begin{bmatrix} 0 & -\mathbf{v}_2 & \mathbf{v}_1 \\ \mathbf{v}_2 & 0 & -\mathbf{v}_0 \\ -\mathbf{v}_1 & \mathbf{v}_0 & 0 \end{bmatrix} \quad (\text{B.0.0.33})$$

$$\begin{bmatrix} \bar{\mathbf{v}}_0 \\ \bar{\mathbf{v}}_1 \\ \bar{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} -\bar{\mathbf{m}}_{12} + \bar{\mathbf{m}}_{21} \\ \bar{\mathbf{m}}_{02} - \bar{\mathbf{m}}_{20} \\ -\bar{\mathbf{m}}_{01} + \bar{\mathbf{m}}_{10} \end{bmatrix} \quad (\text{B.0.0.34})$$

Spatial dyad: $\text{st2sd}()$:

$$\begin{bmatrix} \mathbf{n}_{11} & \mathbf{n}_{12} \\ \mathbf{n}_{21} & \mathbf{n}_{22} \end{bmatrix} = \text{sd}_{\text{st2sd}}\left(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}\right) = \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ -\mathbf{E}\mathbf{r}_\times & \mathbf{E} \end{bmatrix} \quad (\text{B.0.0.35})$$

$$\begin{bmatrix} \bar{\mathbf{E}} \\ \bar{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{n}}_{11}^T + \bar{\mathbf{n}}_{22}^T - \mathbf{r}_\times \bar{\mathbf{n}}_{21} \\ -(\mathbf{E}\bar{\mathbf{n}}_{21})_\times \end{bmatrix} \quad (\text{B.0.0.36})$$

Spatial dyad: shift():

$$\mathbf{n} = sd_{shift}\left(\begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix}, \begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}\right) = \mathbf{a}^T \mathbf{b} \mathbf{a} (st2sd(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}), \begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix}) \quad (\text{B.0.0.37})$$

Adjoint of shift() can be decomposed into the adjoints of $\mathbf{A}^T \mathbf{B} \mathbf{A}()$ and $st2sd()$.

Quaternion: mul_vec():

$$\begin{bmatrix} \mathbf{q}_1.x \\ \mathbf{q}_1.y \\ \mathbf{q}_1.z \\ \mathbf{q}_1.w \end{bmatrix} = qt_{mul_vec}\left(\begin{bmatrix} \mathbf{q}_2.x \\ \mathbf{q}_2.y \\ \mathbf{q}_2.z \\ \mathbf{q}_2.w \end{bmatrix}, \begin{bmatrix} \mathbf{v}.x \\ \mathbf{v}.y \\ \mathbf{v}.z \end{bmatrix}\right) = \begin{bmatrix} \mathbf{q}_2.w * \mathbf{v}.x + \mathbf{q}_2.y * \mathbf{v}.z - \mathbf{q}_2.z * \mathbf{v}.y \\ \mathbf{q}_2.w * \mathbf{v}.y + \mathbf{q}_2.z * \mathbf{v}.x - \mathbf{q}_2.x * \mathbf{v}.z \\ \mathbf{q}_2.w * \mathbf{v}.z + \mathbf{q}_2.x * \mathbf{v}.y - \mathbf{q}_2.y * \mathbf{v}.x \\ -\mathbf{q}_2.x * \mathbf{v}.x - \mathbf{q}_2.y * \mathbf{v}.y - \mathbf{q}_2.z * \mathbf{v}.z \end{bmatrix} \quad (\text{B.0.0.38})$$

$$\begin{bmatrix} \bar{\mathbf{q}}_2.x \\ \bar{\mathbf{q}}_2.y \\ \bar{\mathbf{q}}_2.z \\ \bar{\mathbf{q}}_2.w \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{q}}_1.x * q.w - \bar{\mathbf{q}}_1.y * q.z + \bar{\mathbf{q}}_1.z * q.y - \bar{\mathbf{q}}_1.w * q.x \\ \bar{\mathbf{q}}_1.x * q.z + \bar{\mathbf{q}}_1.y * q.w - \bar{\mathbf{q}}_1.z * q.x - \bar{\mathbf{q}}_1.w * q.y \\ -\bar{\mathbf{q}}_1.x * q.y + \bar{\mathbf{q}}_1.y * q.x + \bar{\mathbf{q}}_1.z * q.w - \bar{\mathbf{q}}_1.w * q.z \end{bmatrix} \quad (\text{B.0.0.39})$$

$$\begin{bmatrix} \mathbf{v}.x \\ \mathbf{v}.y \\ \mathbf{v}.z \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{q}}_1.x * \mathbf{q}_2.w - \bar{\mathbf{q}}_1.y * \mathbf{q}_2.z + \bar{\mathbf{q}}_1.z * \mathbf{q}_2.y - \bar{\mathbf{q}}_1.w * \mathbf{q}_2.x \\ \bar{\mathbf{q}}_1.x * \mathbf{q}_2.z + \bar{\mathbf{q}}_1.y * \mathbf{q}_2.w - \bar{\mathbf{q}}_1.z * \mathbf{q}_2.x - \bar{\mathbf{q}}_1.w * \mathbf{q}_2.y \\ -\bar{\mathbf{q}}_1.x * \mathbf{q}_2.y + \bar{\mathbf{q}}_1.y * \mathbf{q}_2.x + \bar{\mathbf{q}}_1.z * \mathbf{q}_2.w - \bar{\mathbf{q}}_1.w * \mathbf{q}_2.z \end{bmatrix} \quad (\text{B.0.0.40})$$

Quaternion: mul_qt():

$$\begin{bmatrix} \mathbf{q}.x \\ \mathbf{q}.y \\ \mathbf{q}.z \\ \mathbf{q}.w \end{bmatrix} = qt_{mul_qt} \left(\begin{bmatrix} \mathbf{q}_1.x \\ \mathbf{q}_1.y \\ \mathbf{q}_1.z \\ \mathbf{q}_1.w \end{bmatrix}, \begin{bmatrix} \mathbf{q}_2.x \\ \mathbf{q}_2.y \\ \mathbf{q}_2.z \\ \mathbf{q}_2.w \end{bmatrix} \right) = \begin{bmatrix} \mathbf{q}_1.w * \mathbf{q}_2.x + \mathbf{q}_1.x * \mathbf{q}_2.w + \mathbf{q}_1.y * \mathbf{q}_2.z - \mathbf{q}_1.z * \mathbf{q}_2.y \\ \mathbf{q}_1.w * \mathbf{q}_2.y + \mathbf{q}_1.y * \mathbf{q}_2.w + \mathbf{q}_1.z * \mathbf{q}_2.x - \mathbf{q}_1.x * \mathbf{q}_2.z \\ \mathbf{q}_1.w * \mathbf{q}_2.z + \mathbf{q}_1.z * \mathbf{q}_2.w + \mathbf{q}_1.x * \mathbf{q}_2.y - \mathbf{q}_1.y * \mathbf{q}_2.x \\ \mathbf{q}_1.w * \mathbf{q}_2.w - \mathbf{q}_1.x * \mathbf{q}_2.x - \mathbf{q}_1.y * \mathbf{q}_2.y - \mathbf{q}_1.z * \mathbf{q}_2.z \end{bmatrix} \quad (\text{B.0.0.41})$$

$$\begin{bmatrix} \bar{\mathbf{q}}_1.x \\ \bar{\mathbf{q}}_1.y \\ \bar{\mathbf{q}}_1.z \\ \bar{\mathbf{q}}_1.w \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{q}}.x * \mathbf{q}_2.w - \bar{\mathbf{q}}.y * \mathbf{q}_2.z + \bar{\mathbf{q}}.z * \mathbf{q}_2.y - \bar{\mathbf{q}}.w * \mathbf{q}_2.x \\ \bar{\mathbf{q}}.x * \mathbf{q}_2.z + \bar{\mathbf{q}}.y * \mathbf{q}_2.w - \bar{\mathbf{q}}.z * \mathbf{q}_2.x - \bar{\mathbf{q}}.w * \mathbf{q}_2.y \\ -\bar{\mathbf{q}}.x * \mathbf{q}_2.y + \bar{\mathbf{q}}.y * \mathbf{q}_2.x + \bar{\mathbf{q}}.z * \mathbf{q}_2.w - \bar{\mathbf{q}}.w * \mathbf{q}_2.z \\ \bar{\mathbf{q}}.x * \mathbf{q}_2.x + \bar{\mathbf{q}}.y * \mathbf{q}_2.y + \bar{\mathbf{q}}.z * \mathbf{q}_2.z + \bar{\mathbf{q}}.w * \mathbf{q}_2.w \end{bmatrix} \quad (\text{B.0.0.42})$$

$$\begin{bmatrix} \bar{\mathbf{q}}_2.x \\ \bar{\mathbf{q}}_2.y \\ \bar{\mathbf{q}}_2.z \\ \bar{\mathbf{q}}_2.w \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{q}}.x * \mathbf{q}_1.w + \bar{\mathbf{q}}.y * \mathbf{q}_1.z - \bar{\mathbf{q}}.z * \mathbf{q}_1.y - \bar{\mathbf{q}}.w * \mathbf{q}_1.x \\ -\bar{\mathbf{q}}.x * \mathbf{q}_1.z + \bar{\mathbf{q}}.y * \mathbf{q}_1.w + \bar{\mathbf{q}}.z * \mathbf{q}_1.x - \bar{\mathbf{q}}.w * \mathbf{q}_1.y \\ \bar{\mathbf{q}}.x * \mathbf{q}_1.y - \bar{\mathbf{q}}.y * \mathbf{q}_1.x + \bar{\mathbf{q}}.z * \mathbf{q}_1.w - \bar{\mathbf{q}}.w * \mathbf{q}_1.z \\ \bar{\mathbf{q}}.x * \mathbf{q}_1.x + \bar{\mathbf{q}}.y * \mathbf{q}_1.y + \bar{\mathbf{q}}.z * \mathbf{q}_1.z + \bar{\mathbf{q}}.w * \mathbf{q}_1.w \end{bmatrix} \quad (\text{B.0.0.43})$$

Please refer to our code for more details.

Appendix C: Differentiable Simulation of Soft Multi-body Systems

Proof for Proposition 4.4.1

Below we prove Proposition 4.4.1. For ease of analysis, we assume the soft body has only one tetrahedron.

According to our assumption, we have:

$$\frac{\mathbf{M}^x}{h^2} = \begin{bmatrix} \frac{\rho V}{4h^2} & & & \\ & \frac{\rho V}{4h^2} & & \\ & & \frac{\rho V}{4h^2} & \\ & & & \frac{\rho V}{4h^2} \end{bmatrix} \quad \mathbf{L}^x = \mu V \mathbf{A}^{x\top} \mathbf{A}^x \quad (\text{C.0.0.1})$$

$$\mathbf{A}^x = \begin{bmatrix} -\mathbf{P}^{-\top} \mathbf{s} & \mathbf{P}^{-\top} \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} \mathbf{q}_1 - \mathbf{q}_0 & \mathbf{q}_2 - \mathbf{q}_0 & \mathbf{q}_3 - \mathbf{q}_0 \end{bmatrix} \quad \mathbf{s} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^\top \quad (\text{C.0.0.2})$$

where $*_x$ means the submatrix of the original one after extracting the rows and columns that represents the x axis (we omit the other axes because the coefficients are the same), ρ is the density, V the tetrahedron volume, μ the material stiffness, $\mathbf{P}$ the displacement matrix for the tetrahedron, and $\mathbf{p}_i$ the coordinates of the i^{th} vertex. Here, $\mathbf{A}_x$ can be regarded as the spatial differential operator, which is usually used to compute the deformation gradient of the soft body.

Assuming fixed forces of $\mathbf{k}$ and ξ in Eq. 4.4.0.1, the iterative method becomes the Jacobi method, whose convergence is determined by the spectral radius $\rho(\mathbf{M}^{-1}\mathbf{L})$. We relax this condition to obtain a sufficient condition: if the diagonals of $\mathbf{M}$ is larger than the row sums of $\mathbf{L}$, the Jacobi method is guaranteed to converge.

By definition we have

$$\frac{\rho V}{4h^2} > \sum_j |\mathbf{L}_{ij}^x| \quad (\text{C.0.0.3})$$

Since $\mathbf{L}_{ij} = \mu V \langle \mathbf{A}_{*i}, \mathbf{A}_{*j} \rangle$, we have

$$\frac{\rho}{4h^2\mu} > \sum_j |\langle \mathbf{a}_i, \mathbf{a}_j \rangle| \geq 6\|\mathbf{P}^{-1}\mathbf{P}^{-\top}\|_{\infty} \quad (\text{C.0.0.4})$$

since $\mathbf{A}^{x\top}\mathbf{A}^x = \begin{bmatrix} \mathbf{s}^{\top}\mathbf{P}^{-1}\mathbf{P}^{-\top}\mathbf{s} & -\mathbf{s}\mathbf{P}^{-1}\mathbf{P}^{-\top} \\ -\mathbf{P}^{-1}\mathbf{P}^{-\top}\mathbf{s} & \mathbf{P}^{-1}\mathbf{P}^{-\top} \end{bmatrix}$. Further reducing the right hand side yields:

$$\frac{\rho}{24h^2\mu} > \|\mathbf{P}^{-1}\mathbf{P}^{-\top}\|_{\infty} \geq \frac{1}{\sqrt{3}}\|\mathbf{P}^{-1}\mathbf{P}^{-\top}\|_2 = \frac{1}{\sqrt{3}}\lambda_{\min}^{-1}(\mathbf{P}^{\top}\mathbf{P}) \quad (\text{C.0.0.5})$$

where $\lambda_{\min}$ denotes the minimum eigenvalue of the matrix. It is then straightforward to obtain the inequality with respect to the edge lengths of the tetrahedron:

$$\lambda_{\min}(\mathbf{P}^{\top}\mathbf{P}) \leq \frac{1}{3}\text{tr } \mathbf{P}^{\top}\mathbf{P} = \frac{1}{3} \sum_{k=1}^3 \|\mathbf{q}_k - \mathbf{q}_0\|_2^2 \quad (\text{C.0.0.6})$$

To sum up, we reach a condition that constrains the upper bound of h :

$$h^2 < \frac{\rho}{24\sqrt{3}\mu \sum_{k=1}^3 \|\mathbf{q}_k - \mathbf{q}_0\|_2^2} \quad (\text{C.0.0.7})$$

In fact, it is easy to extend the results to a soft body system with multiple tetrahedrons. All we need to do is to sum up the $\mathbf{L}$ matrix for all individual elements in Eq. C.0.0.3 while increasing the mass matrix by a small constant factor (from $\rho/4$ to $O(1)\rho$, so we omit it for convenience). The upper bound of h of the original method becomes even smaller:

$$h^2 < \frac{\rho}{24\sqrt{3}T\mu \sum_{k=1}^3 \|\mathbf{q}_k - \mathbf{q}_0\|_2^2} \quad (\text{C.0.0.8})$$

where T is the number of tetrahedrons. We assume the soft body has T identical tetrahedrons for analysis convenience. For general cases, one can simply replace $T \sum_{k=1}^3 \|\mathbf{q}_k - \mathbf{q}_0\|_2^2$ with $\sum_i \sum_{k=1}^3 \|\mathbf{q}_{i,k} - \mathbf{q}_{i,0}\|_2^2$ where i is the index of tetrahedrons. Using the setting in our experiments, where $T \approx 1000$, $\mu \approx 3 \times 10^5$, $\|\mathbf{q}_k - \mathbf{q}_0\|_2 \approx 10^{-2}$, and $\rho \approx 1$, we would need to set $h < 1/1934$ in order to ensure convergence.

Convergence Proof for Eq. 4.4.0.3 in Splitting Scheme

Below we prove that Eq. 4.4.0.3 is guaranteed to converge. Having the same assumption as Appendix C, the convergence now is determined by the spectral radius $\rho((\mathbf{M} + \mathbf{D})^{-1}(\mathbf{L} - \mathbf{D}))$. We relax this condition to obtain a sufficient condition: if $\mathbf{M} + \mathbf{L}$ is strictly row diagonally

dominating, the Jacobi method is guaranteed to converge.:

$$\frac{\rho V}{4h^2} + |\mathbf{L}_{ii}^x| > \sum_{i \neq j} |\mathbf{L}_{ij}^x| \quad (\text{C.0.0.9})$$

for all $1 \leq i \leq 4$, yielded by the definition of row diagonally dominance. Since $\mathbf{L}_{ij} = \mu V \langle \mathbf{A}_{*i}, \mathbf{A}_{*j} \rangle$, we have

$$\frac{\rho}{4h^2\mu} > \sum_{i \neq j} |\langle \mathbf{a}_i, \mathbf{a}_j \rangle| - |\langle \mathbf{a}_i, \mathbf{a}_i \rangle| \geq 0 \quad (\text{C.0.0.10})$$

given that $\sum_j \langle \mathbf{a}_i, \mathbf{a}_j \rangle = 0$ since the row sum of $\mathbf{A}^x$ is $\mathbf{0}$. This completes the proof.

To extend this conclusion to multiple tetrahedrons, we use the same method discussed before: we sum up the $\mathbf{L}$ matrix for all individual elements in Eq. C.0.0.9. Since the right hand side is 0, summing up the inequalities will not change the conclusion; the convergence is still guaranteed.

Joints

We give two examples here: 1-DoF rotational joint and prismatic joint.

Rotational joint. This joint is characterized by a rotation axis $\mathbf{n}$ and the angle θ . Its transformation

matrix and the Jacobian are:

$$\mathbf{A}^r = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \quad \frac{\partial \mathbf{A}^r}{\partial \theta} = \begin{bmatrix} \frac{\partial \mathbf{R}}{\partial \theta} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \quad (\text{C.0.0.11})$$

$$\mathbf{R} = \cos \theta \cdot \mathbf{I} + \sin \theta [\mathbf{n}]_{\times} + (1 - \cos \theta) \mathbf{n} \mathbf{n}^{\top} \quad (\text{C.0.0.12})$$

$$\frac{\partial \mathbf{R}}{\partial \theta} = -\sin \theta \cdot \mathbf{I} + \cos \theta [\mathbf{n}]_{\times} + \sin \theta \mathbf{n} \mathbf{n}^{\top} \quad (\text{C.0.0.13})$$

The local update of the rotational joint is given by:

$$\theta^{i+1} = \arctan(\sin \theta^i + \cos \theta^i \Delta \theta^i, \cos \theta^i - \sin \theta^i \Delta \theta^i) \quad (\text{C.0.0.14})$$

Prismatic joint. This joint is characterized by a prismatic axis $\mathbf{u}$ and the scale l . Its transformation matrix and the Jacobian are:

$$\mathbf{A}^p = \begin{bmatrix} \mathbf{I} & l \mathbf{u} \\ \mathbf{0} & 1 \end{bmatrix} \quad \frac{\partial \mathbf{A}^p}{\partial l} = \begin{bmatrix} \mathbf{0} & \mathbf{u} \\ \mathbf{0} & 0 \end{bmatrix} \quad (\text{C.0.0.15})$$

$$(\text{C.0.0.16})$$

The local update of the prismatic joint is simply addition:

$$l^{i+1} = l^i + \Delta l^i \quad (\text{C.0.0.17})$$

Actuators

Pneumatic actuator. We use co-rotational elastic strain energy model for tetrahedral cells. For a pneumatic cell with activation level a , the energy is computed as

$$\Psi_{pneumatic}(\mathbf{F}, a) = \frac{k_p}{2} \|\mathbf{F} - \mathbf{R}(a)\|^2 \quad (\text{C.0.0.18})$$

To compute $\mathbf{R}(a)$, we first perform SVD decomposition on the deformation gradient $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. Then $\mathbf{R}(a)$ can be written as $\mathbf{R}(a) = \mathbf{U}\mathbf{\Sigma}^*\mathbf{V}^T$, where $\mathbf{\Sigma}^* = \mathbf{D} + \mathbf{\Sigma}$, and $\mathbf{D}$ is computed by

$$\arg \min_{\mathbf{D}} \|\mathbf{D}\|_2^2, s.t. \prod_i (\Sigma_i + D_i) = a \quad (\text{C.0.0.19})$$

Muscle actuator. We use the muscle actuators described in [75]. Muscles are modeled as fibers in the soft bodies, and the forces are computed as $\mathbf{f}_{muscle}(a) = -f_{muscle}(a)\mathbf{m}$, where $a \in [0, 1]$ is the activation level, $\mathbf{m}$ is the direction of fiber. To implement this force, a strain energy model [63] is used, $E_{muscle} = \mathbf{V}_{muscle} \Psi_{muscle}(\mathbf{F}, e)$, where $\Psi_{muscle}(\mathbf{F}, a) = \frac{k_m}{2} \|(1 - r)\mathbf{F}\mathbf{m}\|$, k_m is the stiffness, $r = \frac{1-a}{l}$ is the projection of the cord segment, $l = \|\mathbf{F}\mathbf{m}\|$ is the stretch factor.

Ablation study

Figure C.1 shows the ablation study of skeletons. In the experiment, a Baymax in its T-pose is released above the ground, as shown in Figure C.1(a). In (b), we embed 5 bones inside the Baymax (4 in arms and legs, and 1 in the torso). When the Baymax falls to the ground, we also add torques on its shoulders so it can lift its arms. (c) shows the result of [59], where

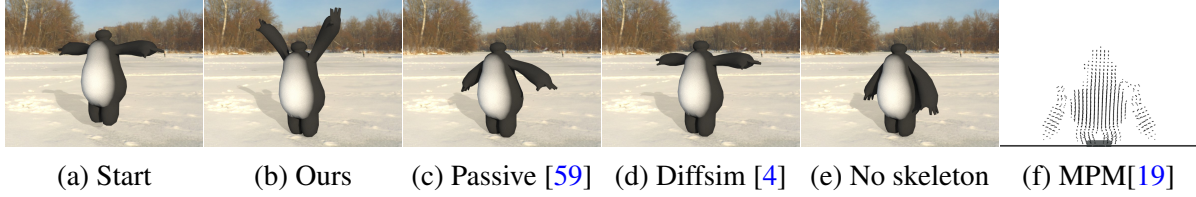


Figure C.1: Ablation study of embedded skeletons. (a) is the initial frame where a Baymax is released from the air and will hit the ground. (b) is a baymax with bones, torques are applied to two shoulder joints so it lifts the arms. (c) is simulated by passive skeleton [59]. (d) Baymax in the differentiable rigid body simulator [4] does not have deformation. Baymax has no skeletons in (e), where the arms are stretched and the legs are collapsed. The MPM particles scatters after the collision in (f).

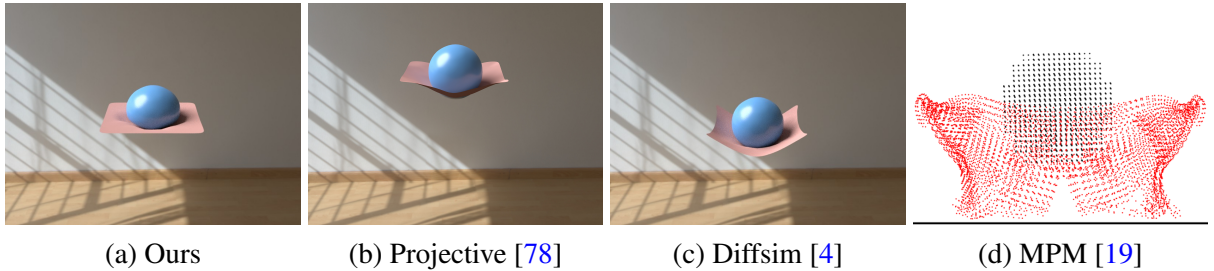


Figure C.2: Ablation study of collision handling scheme. A ball collides with a cloth. Our method in (a) can simulate the deformation of both objects. The dry frictional model for cloth simulates (b), where the solid ball is modeled as rigid body and there is penetration because of the nodal collision handling. The ball is also modeled as a rigid body by [4] in (c). The cloth is teared apart when modeled as MPM particles in (d).

the skeleton is passive and we cannot apply torques. (d) is simulated by another differentiable simulator, DiffSim [4]. It can only simulate cloth and rigid bodies, so Baymax maintains the rest pose. (e) is simulated without a skeleton. Without the support of rigid bones, we notice that its arms are stretched and its legs are compressed. We also run DiffTaichi-MPM in (d), where the arms of the Baymax are detached from the body after collision.

Figure C.2 is the ablation study for different contact handling schemes. A soft ball is released from the air and then collide with a cloth. (a) is our simulation results, where both the ball and the cloth have significant deformation. We run the dry frictional contact algorithm designed for cloth [78] in (b), but the method cannot simulate 3D soft objects, so the ball is rigid. And there is also slight penetration due to the instability of nodal collision handling. (c) is DiffSim [4], which can simulate cloth and rigid body, but the ball is not differentiable. (d) is simulated by DiffTaichi-MPM. The objects can deform, but the elasticity is not realistic and the cloth will break down after the collision.

Appendix D: Editable Neural Geometry and Physics from Monocular Videos

Weight Function

We follow the NeuS [88] paper to set the weight function $w(t)$ in Equation 6.3.1.1. First, a density function $\rho(t)$ is defined as:

$$\rho(t) = \max\left(\frac{-\frac{d\Phi_h}{dt}(s(\mathbf{p}(t)))}{\Phi_h(s(\mathbf{p}(t)))}, 0\right) \quad (\text{D.0.0.1})$$

where $\Phi_h(x) = (1 + e^{-hx})^{-1}$ is the Sigmoid function, and $w(t)$ can be constructed as:

$$w(t) = T(t)\rho(t), \text{ where } T(t) = \exp\left(-\int_0^t \rho(u)du\right). \quad (\text{D.0.0.2})$$

Additional Implementation Details

The Color Network (MLP) consists of 4 dense layers, each with 256 hidden dimensions, while the SDF Network has 8 dense layers of 256 hidden dimensions, the Motion Network has 5 dense layers of 64 hidden dimensions, and the Rigidity Network has 3 dense layers of 32 hidden dimensions.

Similar to NeRF++ [85], we use a separate NeRF network to render far-away points outside of a unit-sphere region of interest. The Motion and Rigidity networks are not applied to those

regions since we assume the background to be static. If we do not disable these networks on the background, we observe significant jittering on those regions.

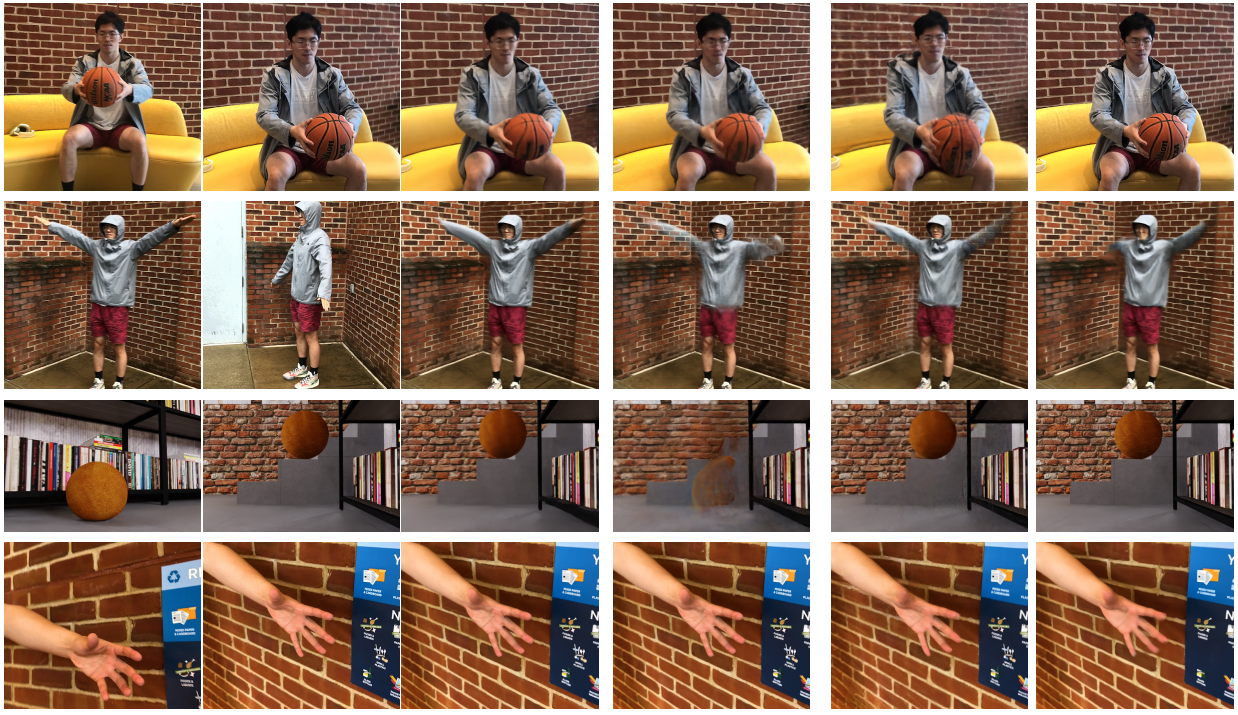
We have also tried enabling and disabling viewing direction as an input to the Color Network. It turns out that disabling viewing direction can result in smoother geometry, as it prevents over-fitting to the viewing direction as discussed in NeRF++. However, we also observed that disabling the viewing direction led to incorrect geometries (like holes) in some cases.

Potential Applications

NeuPhysics enables high-resolution mesh reconstruction from a casual video captured by a smartphone. It would be easy for users to collect their own data to customize virtual body avatars for games and social applications. Accurate facial and body models can also be used in virtual try-on, or even customized suits, furniture, and sports equipment. Moreover, the physically-based, dynamic, photorealistic editing technique could be used in designing, content creation, and broader VR/AR applications.

Other Supplementary Materials

More dynamic visualization results can be found in our supplementary video; we invite the readers to view these videos for visual comparison. A preliminary version of our code is included for reviewing only. We will also open-source a release version in the future for research reproduction with the paper publication.



(a) Input 1 (b) Input 2 (c) Ours (d) NRNeRF [106] (e) D-NeRF [105] (f) NeuS [88]

Figure D.1: **Qualitative evaluation of video reconstruction.** We evaluate four methods on both real and synthetic videos. Our method outperforms other SOTA methods with higher fidelity and less blur.

Video Reconstruction

For the video reconstruction task, we evaluate the methods on a dataset consisting of both real-world captured and synthetic videos. The synthetic data are animated by Blender, and the real-world videos are taken by a smartphone in both indoor and outdoor scenes. Figure D.1 visualizes the datasets as well as the reconstruction videos by different methods. (a, b) are two frames from the input videos where both the scene and the camera positions have changed. (c) is by our method, while (d) NRNeRF [106] (e) D-NeRF [105], and (f) NeuS [88] are SOTA methods for comparison. Note that (d) and (e) are designed for dynamic videos as input, but not designed for geometry reconstruction. (f) learns SDF field during the video reconstruction as ours does, however, it is originally designed for *static* scenes so inconsistency among views would likely undermine its performance.

Appendix E: Dynamic Mesh-Aware Radiance Fields

Runtime and Memory

To break down the time and memory cost of our method, Figure E.3 profiles the simulation and rendering modules as the number of synthetic objects increases. In the test scene, we throw $n \in \{1, 10, 100, 1000, 10000\}$ balls onto a Fox [224] represented by NeRF. The rendering resolution is 900×450 .

As seen in the figure, the simulation and rendering memory stay constant at a low level (0.5 GB and 1.0 GB, respectively) throughout the entire experiment. The run time for both modules scales practically linearly w.r.t. the number of objects.

Moreover, the simulation module is independent of the resolution, and the rendering time scales linearly w.r.t. the number of pixels (see left). This pipeline can run in real time (20 FPS) at 600×300 resolution.

Comparison with density fields

We compare the simulation quality with different collision geometry. In Figure E.5, we drop four balls above a messy counter top [287]. In this simulation scene, (a) models the collision geometry using learned SDF by our NeuS-NGP and (b) directly uses Marching-Cube mesh from instant-NGP density fields (with resolution $256 \times 256 \times 256$). Two of the balls in (b) ‘sink’ into

the countertop, which is unrealistic. In contrast, our SDF-based simulation in (a) can correctly simulate the interaction between the balls and the background objects. More dynamics results can be found in our supplementary video.

Comparison with other NeRF-based simulation.

There are two recent works [269, 271] that can simulate in neural fields. [271] establish a collision model based on the density field. In our simulation, we found that the density fields are usually noisy and inadequate to model a surface well for accurate contact processing. NeuPhysics[269] aims at differentiable simulation and rendering, and it extracts hexahedra mesh from learned SDF [88] and simulates the mesh using [151]. NeuPhysics can only simulate existing NeRF objects instead of synthetic objects. Table E.4 shows that our method runs at least an order of magnitude faster when simulating a bouncing ball.

Light Source Estimation - Implementation Details

Geometry Reconstruction

Given our choice to represent light sources as area lights defined on an explicit mesh, we first need to reconstruct the geometry of the scene. We choose to use MonoSDF [284], which leverages monocular geometric priors (depth and normal estimation) in the neural implicit surface reconstruction process. Importantly, these priors especially help in reconstructing surfaces that have only a low degree of visual texture, which is very common for walls and flat tabletop surfaces. For differentiable surface rendering, having reasonably accurate geometry is critical for achieving good convergence, as noisy geometry will tend to bias the result to bad local minima.

Empirically, MonoSDF achieves sufficiently accurate geometry for this purpose.

Once we have an optimized signed distance field, we convert it to an explicit mesh using Marching Cubes, then UV unwrap the mesh to obtain a UV texture map, using SDFStudio [229, 383]. This step also results in a UV texture map whose values are determined by querying the underlying MonoSDF appearance model, which we use as an albedo UV texture map. The UV-unwrapped mesh is the input to the differentiable surface render, described below.

Differentiable Surface Rendering

We implement our differentiable surface rendering optimization using Mitsuba3 [230], which is built on Dr. Jit [285], a just-in-time compiler that is specialized for rendering use cases. Our inverse rendering procedure is generally based on [286], though our specific implementation details are as follows. The UV-unwrapped scene mesh is assigned a Principled BSDF material [384], as well as an emission UV texture map, which constitute the inputs to the differentiable renderer. The emission UV texture map, $T_{emission}$, with dimensions (h_{tx}, w_{tx}, c_{tx}) , is the variable that we are interested in optimizing. It is initialized uniformly with near-zero values. The other BSDF parameters remain fixed (albedo texture map, specular transmission map with default value 1.0, and roughness map with default value 0.5). For rendering, we use a Path-Replay Backpropagation (PRB) integrator [385], and limit the light transport simulation to a maximum depth of 3 (2 bounces). This stands in contrast to differentiable rasterization approaches to inverse rendering, which are limited to a maximum depth of 2 (1 bounce), and are therefore unable to account for indirect lighting in the optimization. We also directly render emitters (as opposed to only emitting light onto surfaces without being directly visible).

Given a set of ground-truth 32-bit HDR images $\{I_1, \dots, I_n\}$ with dimensions (h_{im}, w_{im}, c_{im}) and known camera poses, our primary objective is to minimize the rendering loss (mean-squared error) between the ground truth images, and images rendered from identical camera poses. Since the variables that we are optimizing are the emission UV texture map values, we also find that applying uniform L1 regularization to the optimization variable corresponds well to the inductive bias that the majority of the scene does not emit light, and therefore the emission UV texture map should be encouraged to be sparse. For each optimization step $i \in [1, n]$ in a given optimization epoch, our objective is therefore to minimize the following, where α is a tunable hyperparameter:

$$\text{minimize } \frac{1}{(h_{im} * w_{im} * c_{im})} \|I_i - \hat{I}_i\|^2 + \frac{\alpha}{(h_{tx} * w_{tx} * c_{tx})} |T_{emission}|$$

In addition to the rendering loss and L1 regularization in texture space, we also find that periodically clipping low values and boosting large values in the emission texture space encourages convergence toward correct results. The motivation is that periodically during the optimization procedure (according to a tunable cadence, where we assume a default period of every 2 epochs), emission values below a given brightness threshold (also a tunable hyperparameter, with a default value of 0.2), are clipped to 0. By assuming that very dim values are not light sources, we avoid local minima. At the same time, values that are above this threshold are likely to be light sources, and therefore, we boost these values, in order to accelerate the optimization, since extreme emission values in the 32-bit linear color space may be arbitrarily larger than 1. This part of the optimization should be self-correcting: values that are boosted when they shouldn't be will then be brought down in order to satisfy the rendering loss objective. See Figure E.1 for an example of the converged result of optimizing the emission texture map.

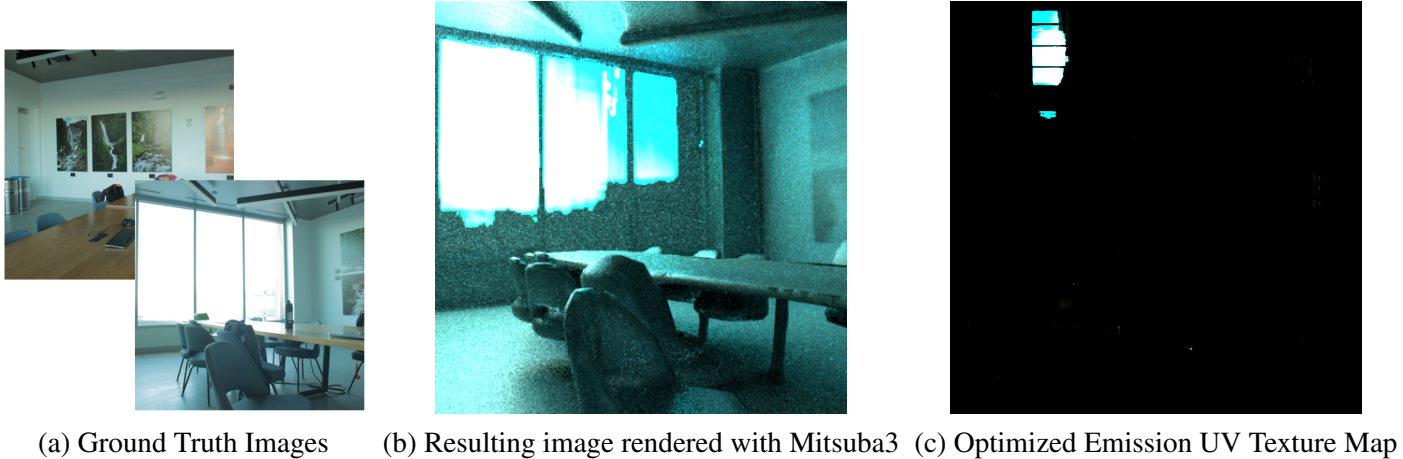


Figure E.1: **Optimizing Emission UV Texture Map.** We optimize the (c) Emission UV Texture Map by minimizing a rendering loss objective between (a) ground truth images and (b) images rendered with Mitsuba3, a differentiable surface rendering engine.

Finally, once the optimization has converged, we post process the mesh by pruning off any triangular faces whose vertices all fall below a threshold, by looking up the brightness of each vertex in the emission UV texture map, according to that vertex’s associated UV coordinates. This results in a greatly reduced mesh that only contains faces that should emit light onto the rest of the scene, and can therefore be used as an area light for computing the shadow pass of our Hybrid Renderer.

HDR Rendering Experiment Details

Data acquisition

To train the HDR NeRF, we first construct a set of 32-bit HDR images of a scene, where each image is analogous to a single LDR image in the baseline NeRF formulation. To create a single one of these HDR images, we shoot a bracketed series of LDR exposures that consists of several images (7 in our experiments) captured from a fixed camera pose. We use a single Canon 5D MKIII with a 28mm lens, and set the camera to record images in 8-bit JPEG format.

Specifically, the camera is mounted to a tripod to minimize any minute difference in camera pose between each of the exposures in the bracket, and to minimize any optical differences between the images (e.g. depth of field, bloom, etc.) apart from raw exposure value, we only adjust the exposure duration (i.e. shutter speed) between images, while the aperture remains fixed. The *relative* difference in exposure between any consecutive images in the bracket is kept constant, though the magnitude of this difference depends on the degrees of dynamic range of the scene being captured; ideally, within the range of exposures from darkest to brightest, every pixel of the image should be “well-exposed” in at least one of the images. We generally captured 7 images in the bracketed exposure, with 1-2 stops difference between each image.

Data preprocessing

After capturing the scene, we preprocess the data using OpenCV. For each bracketed exposure consisting of n images taken from the same camera pose with different exposures, we align the images to ensure the best possible quality of the HDR image result, as even slight perturbations in the camera pose may cause degradation. Then, we select a single bracketed series that is “representative” of the lighting conditions of the full set, roughly covering the upper and lower limits of the overall dynamic range. The representative bracketed series is used to recover the camera’s nonlinear tone-mapping function, which we will refer to as the Camera Response Function (CRF) [Figure E.2] using the method from [293].

To improve computational efficiency, we recover the CRF from only one representative image, although the linear system that must be solved could theoretically incorporate pixels from *all* of the bracketed series as constraints. By applying the recovered CRF uniformly to all series,

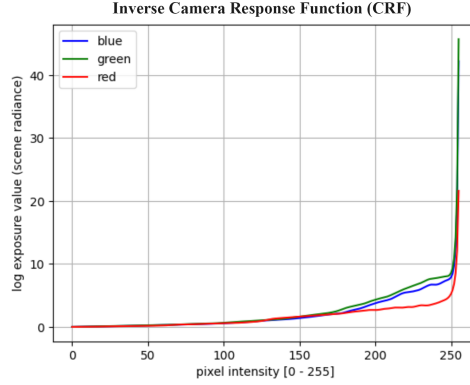


Figure E.2: **Inverse Camera Response Function (CRF)** showing the recovered nonlinear function mapping from sRGB 8-bit pixel color to linear 32-bit radiance.

we achieve consistent levels across all images. After applying the CRF to all bracketed series, we downsample each resulting 32-bit image by a factor of $4x$. Finally, for numerical stability during downstream training, we normalize all of the images by linearly scaling radiance values to the range of $[0, 255]$. While this may seem similar to reducing the HDR result back to LDR, all values are scaled uniformly so the *relative* differences between values are preserved, and they are **not** quantized to integer values. The resulting HDR images are then stored as EXR files.

Note that for performing camera pose estimation using COLMAP, as is widely done in the NeRF literature, we only use a single 8-bit LDR image from each bracketed series. Similarly, when training an LDR model for comparison against its HDR counterpart, we use a single image from each bracketed series, where all of the singleton images have the same exposure duration.

HDR NeRF Implementation Details

Training the HDR NeRF is largely the same as training the baseline (LDR) NeRF. In fact, the Instant-NGP implementation already supports training with 32-bit HDR images in EXR format. We simply need to toggle flags ‘is_hdr’ in Instant-NGP’s data loader and ‘linear_colors’ in

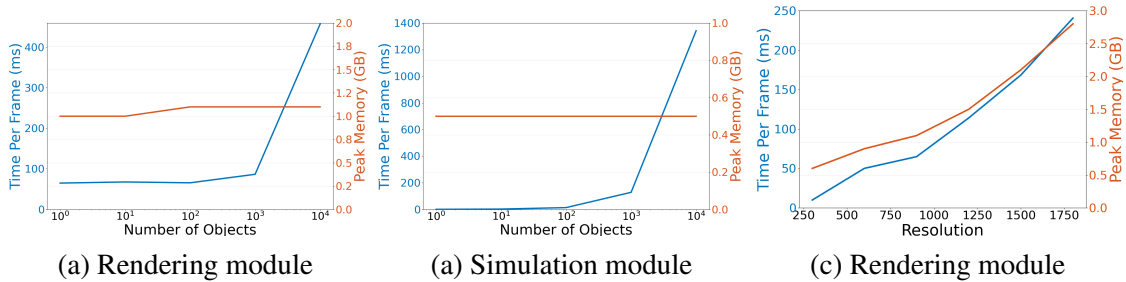


Figure E.3: **Scaling the number of the inserted objects and image resolution.** When we increase the number of simulated objects from 1 to 10^4 , the peak simulation memory and rendering memory stay nearly constant. The (a) simulation time scales linearly w.r.t. the number of objects, while the (b) rendering time can increase when the number of the inserted meshes is too high. The (c) rendering time and memory usage in rendering modules scales linearly w.r.t. the number of pixels.

the GUI. Therefore, training is supervised purely in (scaled) linear color space, and correspondingly, the radiance component of the neural field is defined in linear color space. Optionally, during rendering only, we may choose to apply a tonemapping function uniformly to all of the accumulated pixel radiance values to obtain the final image in sRGB color space.



	Rendering	Simulation
Ours	~ 0.1 s	~ 0.02 s
NeuPhysics	~ 5 s	~ 0.5 s

Figure E.4: **Run time comparison with NeuPhysics [269].** We simulate a ball that bounces around (left). Compared to NeuPhysics, our method supports hybrid NeRF and mesh rendering and runs 25x-50x faster.

Details about the Rendering comparison

We conducted a user study where participants were asked to rank the realism, quality, and correctness of rendered results from three methods, including ours, on the following examples: Mirror (main paper Fig. 2), Garden, and Bicycle (main paper Fig. 5). Of 15 participants, 13/15



(a) Using learned SDF to simulate (b) Using mesh from density fields

Figure E.5: **Simulation Comparison.** Our simulation (a) using learned SDF as collision proxy geometry can correctly handle the contact between the counter and four balls. (b) Using marching-cube mesh directly from the density field, it fails to resolve collision and two of the four balls sink into the counter

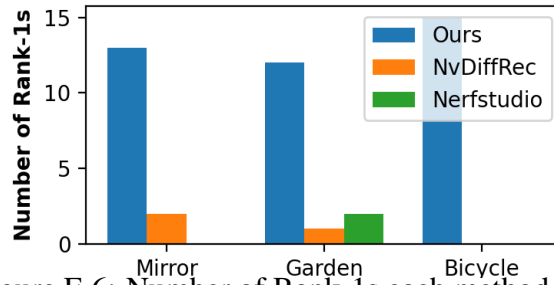


Figure E.6: Number of Rank-1s each method receives.

ranked our results highest for Mirror, while 12/15 did so for Garden and 15/15 for Bicycle (see Fig. E.6). Binomial tests indicate that our images are statistically superior in all three scenes, with significant results at a significance level of 0.05. The corresponding p-values were found to be 0.035, 0.007, and 6×10^{-5} , respectively. We will include the quantitative results in the revised paper.

We also consider metrics such as FID and CLIP [5] scores. However, FID requires a ground truth distribution we cannot access. CLIP is a large language/vision model, but its scores are sensitive to prompts, and it is likely prone to misinterpreting reflections. Defining a scalable and robust quantitative metric to measure the visual fidelity of generative tasks remains an open problem.

MirrorNeRF designs a convenient capture system utilizing mirrors, but it does not focus on



Figure E.7: Comparison between Luma AI (left) and Ours (right). the rendering problem, thus does not overlap with the scope of our work.

LumaAI recently released a UE plugin in April *after* the submission deadline. They provide binaries but no source code, so we only have a limited understanding of how it works. It is stated on their blog that NeRF cannot be used to simulate (otherwise, users need to add collider objects *manually*). Moreover, when we put a mirror in the Garden, shadows in the mirror reflections look strange. By contrast, our reflections and shadows are superior (see Fig. [E.7](#)).

Appendix F: Physically Consistent Perception of Hand-Object Interactions with Differentiable Priors

Extended Kalman Filter

We have a dynamic system,

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t) + \epsilon_s, \mathbf{z}_t = h(\mathbf{x}_t) + \epsilon_o, \quad (\text{F.0.0.1})$$

where $\mathbf{x}$ is the state, $\mathbf{u}$ is the control, $\mathbf{z}$ is the observation, ϵ_s and ϵ_o are multivariate Gaussian noises with covariance $\mathbf{Q}$ and $\mathbf{R}$ respectively, $f(\cdot), h(\cdot)$ are *differentiable* functions for states and observation, respectively. EKF can predict the state $\hat{\mathbf{x}}_t$ and covariance matrix $\mathbf{P}_t$ given the

observation $\mathbf{z}$. The EKF works in the following steps for each frame,

$$\mathbf{F} = \partial f / \partial \hat{\mathbf{x}}_{t-1} \quad (\text{F.0.0.2})$$

$$\mathbf{x}' = f(\hat{\mathbf{x}}_{t-1}, \mathbf{u}_t) \quad (\text{F.0.0.3})$$

$$\mathbf{H} = \partial h / \partial \mathbf{x}' \quad (\text{F.0.0.4})$$

$$\mathbf{P}' = \mathbf{F}_t \mathbf{P}_{t-1} \mathbf{F}_t^\top + \mathbf{Q} \quad (\text{F.0.0.5})$$

$$\mathbf{y} = \mathbf{z}_t - h(\mathbf{x}') \quad (\text{F.0.0.6})$$

$$\mathbf{S} = \mathbf{H} \mathbf{P}' \mathbf{H}^\top + \mathbf{R} \quad (\text{F.0.0.7})$$

$$\mathbf{K} = \mathbf{P}' \mathbf{H}^\top \mathbf{S}^{-1} \quad (\text{F.0.0.8})$$

$$\hat{\mathbf{x}}_t = \mathbf{x}' + \mathbf{K} \mathbf{y} \quad (\text{F.0.0.9})$$

$$\hat{\mathbf{P}}_t = (\mathbf{I} - \mathbf{K} \mathbf{H}) \mathbf{P}' \quad (\text{F.0.0.10})$$

More detailed derivation can be found in [308]. In our experiments, we use the 3D positions of fingertips as the control input. The observation signal could be the hand pose, object bounding boxes, etc.

Quasistatic Time-stepping

We assume that the relative sliding of the contact points on the object is small. For simplicity, we approximately treat the fingertips as the contact points $\mathbf{v}$. Since the object is rigid, we could treat the point cloud at two adjacent frames under rigid transformation. Therefore, we could use SVD-based rigid point registration [386] to compute the relative rotation and translation.

Table F.1: **Error after contact-based pose refinement.** Our differentiable contact module is used to refine the pose given the contact status and achieves comparable results as ContactOpt [340].

Method	hand joints	hand verts	object verts
ContactOpt	0.0143	0.0137	0.0367
ours	0.0143	0.0136	0.0364

Contact Refinement

Table F.1 show the quantitative comparisons with ContactOpt. It demonstrates that the two methods have comparable performance on the 3D hand and object errors.

Implementation Details

Our differentiable priors are implemented using Pytorch and optimized by Stochastic Gradient Descent (SGD) with learning rate 0.01. The weights of loss terms are $\{\lambda_i\}_{i=1, \dots, 6} = \{0.1, 0.3, 0.005, 0.001, 1, 1\}$. The profiling numbers in Table 8.1 are obtained on a desktop with an Intel(R) Xeon(R) W-2123 CPU @ 3.60GHz and an NVIDIA Titan XP GPU. The EKF is from the torchfilter [387] and the differentiable renderer is SoftRas [342].

Bibliography

- [1] Jonas Degraeve, Michiel Hermans, Joni Dambre, and Francis Wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13, 2019.
- [2] Filipe de Avila Belbute-Peres, Kevin A. Smith, Kelsey R. Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In *Neural Information Processing Systems*, 2018.
- [3] Junbang Liang, Ming C. Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. In *Neural Information Processing Systems*, 2019.
- [4] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *ICML*, 2020.
- [5] John Ingraham, Adam Riesselman, Chris Sander, and Debora Marks. Learning protein structure with a differentiable simulator. In *ICLR*, 2019.
- [6] 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 *ICRA*, 2019.
- [7] James M. Bern, Pol Banzet, Roi Poranne, and Stelian Coros. Trajectory optimization for cable-driven soft robot locomotion. In *Robotics: Science and Systems (RSS)*, 2019.
- [8] Tetsuya Takahashi, Junbang Liang, Yi-Ling Qiao, and Ming C Lin. Differentiable fluids with solid coupling for learning and control. In *AAAI*, 2021.
- [9] 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 *Neural Information Processing Systems*, 2019.
- [10] Eric Heiden, David Millard, Hejia Zhang, and Gaurav S Sukhatme. Interactive differentiable simulation. *arXiv:1905.10706*, 2019.
- [11] Kun Wang, Mridul Aanjaneya, and Kostas E. Bekris. A first principles approach for data-efficient system identification of spring-rod systems via differentiable physics engines. In *LADC*, 2020.

- [12] Jatavallabhula Krishna Murthy, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, et al. gradSim: Differentiable simulation for system identification and visuomotor control. In *ICLR*, 2021.
- [13] Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B. Tenenbaum, and Chuang Gan. PlasticineLab: A soft-body manipulation benchmark with differentiable physics. In *ICLR*, 2021.
- [14] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems (RSS)*, 2018.
- [15] Changkyu Song and Abdeslam Boularias. Identifying mechanical models of unknown objects with differentiable physics simulations. In *L4DC*, 2020.
- [16] Changkyu Song and Abdeslam Boularias. Learning to slide unknown objects with differentiable physics simulations. In *Robotics: Science and Systems (RSS)*, 2020.
- [17] Jan Drgona, Aaron Tuor, and Draguna Vrabie. Learning constrained adaptive differentiable predictive control policies with guarantees, 2020.
- [18] Sirui Chen, Keenon Werling, Albert Wu, and C Karen Liu. Real-time model predictive control and system identification using differentiable simulation. *IEEE Robotics and Automation Letters*, 8(1):312–319, 2022.
- [19] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable programming for physical simulation. In *ICLR*, 2020.
- [20] Samuel S. Schoenholz and Ekin D. Cubuk. JAX M.D.: End-to-end differentiable, hardware accelerated, molecular dynamics in pure python. In *Neural Information Processing Systems*, 2020.
- [21] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necoala, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018. URL <http://github.com/google/jax>.
- [22] Erwin Coumans et al. Tiny differentiable simulator. <https://github.com/google-research/tiny-differentiable-simulator>, 2020.
- [23] Bradley M. Bell et al. CppAD: C++ algorithmic differentiation. <https://projects.coin-or.org/CppAD>, 2018.
- [24] Sameer Agarwal, Keir Mierle, et al. Ceres solver. <http://ceres-solver.org>, 2010.

- [25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019.
- [26] Juntang Zhuang, Nicha C. Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James S. Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ODE. In *ICML*, 2020.
- [27] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv:1604.06174*, 2016.
- [28] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. In *Neural Information Processing Systems*, 2016.
- [29] Marisa Kirisame, Steven Lyubomirsky, Altan Haan, Jennifer Brennan, Mike He, Jared Roesch, Tianqi Chen, and Zachary Tatlock. Dynamic tensor rematerialization. In *ICLR*, 2021.
- [30] Aashaka Shah, Chao-Yuan Wu, Jayashree Mohan, Vijay Chidambaram, and Philipp Krähenbühl. Memory optimization for deep networks. In *ICLR*, 2021.
- [31] Antoine McNamara, Adrien Treuille, Zoran Popovic, and Jos Stam. Fluid control using the adjoint method. *ACM Transactions on Graphics (TOG)*, 23(3), 2004.
- [32] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control PDEs with differentiable physics. In *ICLR*, 2020.
- [33] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative backpropagation: An adjoint method for lightning-fast differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(4), 2020.
- [34] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Neural Information Processing Systems*, 2018.
- [35] 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 Transactions on Graphics (TOG)*, 39(6), 2020.
- [36] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Neural Information Processing Systems*, 2016.
- [37] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. In *ICLR*, 2017.
- [38] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Fei-Fei Li, Josh Tenenbaum, and Daniel L. Yamins. Flexible neural representation for physics prediction. In *Neural Information Processing Systems*, 2018.

- [39] Connor Schenck and Dieter Fox. SPNets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning (CoRL)*, 2018.
- [40] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *ICML*, 2018.
- [41] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *ICML*, 2020.
- [42] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *ICRA*, 2019.
- [43] Filipe de Avila Belbute-Peres, Thomas D. Economou, and J. Zico Kolter. Combining differentiable PDE solvers and graph neural networks for fluid flow prediction. In *ICML*, 2020.
- [44] Benjamin Ummerhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *ICLR*, 2020.
- [45] Nils Wandel, Michael Weinmann, and Reinhard Klein. Learning incompressible fluid dynamics from scratch – towards fast, differentiable fluid models that generalize. In *ICLR*, 2021.
- [46] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *ICLR*, 2021.
- [47] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Saucedo Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Neural Information Processing Systems*, 2017.
- [48] Brandon M. Anderson, Truong-Son Hy, and Risi Kondor. Cormorant: Covariant molecular neural networks. In *Neural Information Processing Systems*, 2019.
- [49] Alexander Bogatskiy, Brandon M. Anderson, Jan T. Offermann, Marwah Roussi, David W. Miller, and Risi Kondor. Lorentz group equivariant neural network for particle physics. In *ICML*, 2020.
- [50] Zhengdao Chen, Jianyu Zhang, Martín Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. In *ICLR*, 2020.
- [51] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR Workshops*, 2020.
- [52] Filip Ilievski, Aaron D Mazzeo, Robert F Shepherd, Xin Chen, and George M Whitesides. Soft robotics for chemists. *Angewandte Chemie*, 123(8), 2011.

- [53] Thomas George Thuruthel, Yasmin Ansari, Egidio Falotico, and Cecilia Laschi. Control strategies for soft robotic manipulators: A survey. *Soft Robotics*, 5(2):149–163, 2018.
- [54] Miles Macklin. Simulation for learning and robotics, numerical methods for contact, deformation, and identification. In *PhD Thesis*, 2020.
- [55] Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. Deformable objects alive! *ACM Transactions on Graphics (TOG)*, 31(4), 2012.
- [56] Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. Simulation and control of skeleton-driven soft body characters. *ACM Transactions on Graphics (TOG)*, 32(6), 2013.
- [57] Junggon Kim and Nancy S Pollard. Direct control of simulated nonhuman characters. *IEEE Computer Graphics and Applications*, 31(4):56–65, 2011.
- [58] Nico Galoppo, Miguel A Otaduy, Serhat Tekin, Markus Gross, and Ming C Lin. Soft articulated characters with fast contact handling. In *Computer Graphics Forum*, volume 26, 2007.
- [59] Jing Li, Tiantian Liu, and Ladislav Kavan. Fast simulation of deformable characters with articulated skeletons in projective dynamics. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2019.
- [60] Yifeng Jiang, Tom Van Wouwe, Friedl De Groote, and C Karen Liu. Synthesis of biologically realistic human motion using joint torque actuation. *ACM Transactions On Graphics (TOG)*, 38(4), 2019.
- [61] Sumit Jain and C Karen Liu. Controlling physics-based characters using soft contacts. In *SIGGRAPH Asia*, 2011.
- [62] Yunbo Zhang, Wenhao Yu, C Karen Liu, Charlie Kemp, and Greg Turk. Learning to manipulate amorphous materials. *ACM Transactions on Graphics (TOG)*, 39(6), 2020.
- [63] Seunghwan Lee, Ri Yu, Jungnam Park, Mridul Aanjaneya, Eftychios Sifakis, and Jehee Lee. Dexterous manipulation and control with volumetric muscles. *ACM Transactions on Graphics (TOG)*, 37(4), 2018.
- [64] Baptiste Angles, Daniel Rebain, Miles Macklin, Brian Wyvill, Loic Barthe, JP Lewis, Javier Von Der Pahlen, Shahram Izadi, Julien Valentin, Sofien Bouaziz, et al. Viper: Volume invariant position-based elastic rods. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2(2):1–26, 2019.
- [65] Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. Scalable muscle-actuated human simulation and control. *ACM Transactions On Graphics (TOG)*, 38(4), 2019.
- [66] 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 Transactions on Graphics (TOG)*, 2011.

- [67] Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Transactions on Graphics (TOG)*, 37(2), 2018.
- [68] Jonathan Hiller and Hod Lipson. Dynamic simulation of soft multimaterial 3D-printed objects. *Soft Robotics*, 1(1):88–101, 2014.
- [69] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *SIGGRAPH*, 2002.
- [70] Torsten Hädrich, Bedrich Benes, Oliver Deussen, and Sören Pirk. Interactive modeling and authoring of climbing plants. In *Computer Graphics Forum*, volume 36, pages 49–61. Wiley Online Library, 2017.
- [71] 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 Transactions on Graphics (TOG)*, 2020.
- [72] Minchen Li, Danny M Kaufman, and Chenfanfu Jiang. Codimensional incremental potential contact. *arXiv:2012.04457*, 2020.
- [73] Jie Li, Gilles Daviet, Rahul Narain, Florence Bertails-Descoubes, Matthew Overby, George E Brown, and Laurence Boissieux. An implicit frictional contact solver for adaptive cloth simulation. *ACM Transactions on Graphics (TOG)*, 37(4), 2018.
- [74] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Transactions on Graphics (TOG)*, 33(4), 2014.
- [75] Sehee Min, Jungdam Won, Seunghwan Lee, Jungnam Park, and Jehee Lee. Softcon: simulation and control of soft-bodied animals with biomimetic actuators. *ACM Transactions on Graphics (TOG)*, 38(6), 2019.
- [76] Jing Li, Tiantian Liu, and Ladislav Kavan. Soft articulated characters in projective dynamics. *IEEE Transactions on Visualization and Computer Graphics*, 2020.
- [77] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Quasi-Newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics (TOG)*, 36(3), 2017.
- [78] Mickaël Ly, Jean Jouve, Laurence Boissieux, and Florence Bertails-Descoubes. Projective dynamics with dry frictional contact. *ACM Transactions on Graphics (TOG)*, 39(4), 2020.
- [79] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.
- [80] Frank Dellaert and Lin Yen-Chen. Neural volume rendering: Nerf and beyond. *arXiv preprint arXiv:2101.05204*, 2020.

- [81] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.
- [82] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. Nerd: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12684–12694, 2021.
- [83] Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. *Advances in Neural Information Processing Systems*, 34, 2021.
- [84] Kai Zhang, Fujun Luan, Zhengqi Li, and Noah Snavely. Iron: Inverse rendering by optimizing neural sdfs and materials from photometric images. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.
- [85] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020.
- [86] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6), 2019.
- [87] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(4), 2020.
- [88] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *NeurIPS*, 2021.
- [89] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.
- [90] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *International Conference on Computer Vision (ICCV)*, 2021.
- [91] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018.
- [92] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018.

- [93] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7708–7717, 2019.
- [94] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.
- [95] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. In *proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [96] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016.
- [97] Haozhe Xie, Hongxun Yao, Xiaoshuai Sun, Shangchen Zhou, and Shengping Zhang. Pix2vox: Context-aware 3d reconstruction from single and multi-view images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2690–2698, 2019.
- [98] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020.
- [99] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33:2492–2502, 2020.
- [100] Petr Kellnhofer, Lars C Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. Neural lumigraph rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4287–4297, 2021.
- [101] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), dec 2021.
- [102] Benjamin Attal, Eliot Laidlaw, Aaron Gokaslan, Changil Kim, Christian Richardt, James Tompkin, and Matthew O’Toole. Törf: Time-of-flight radiance fields for dynamic scene view synthesis. *Advances in neural information processing systems*, 34, 2021.
- [103] Jiakai Zhang, Xinhang Liu, Xinyi Ye, Fuqiang Zhao, Yanshun Zhang, Minye Wu, Yingliang Zhang, Lan Xu, and Jingyi Yu. Editable free-viewpoint video using a layered neural representation. *ACM Transactions on Graphics (TOG)*, 40(4):1–18, 2021.

- [104] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021.
- [105] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *arXiv preprint arXiv:2011.13961*, 2020.
- [106] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2021.
- [107] Hsiao-yu Chen, Edgar Tretschk, Tuur Stuyck, Petr Kadlec, Ladislav Kavan, Etienne Vouga, and Christoph Lassner. Virtual elastic objects. *arXiv preprint arXiv:2201.04623*, 2022.
- [108] Yilun Du, Yanan Zhang, Hong-Xing Yu, Joshua B. Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.
- [109] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5712–5721, 2021.
- [110] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9421–9431, 2021.
- [111] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.
- [112] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 2014.
- [113] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. AK Peters/CRC Press, 2010.
- [114] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics (TOG)*, 2002.
- [115] David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. Robust treatment of simultaneous collisions. *ACM Transactions on Graphics (TOG)*, 2008.
- [116] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017.
- [117] Justin Carpentier and Nicolas Mansard. Analytical derivatives of rigid body dynamics algorithms. In *Robotics: Science and Systems (RSS)*, 2018.

- [118] David Millard, Eric Heiden, Shubham Agrawal, and Gaurav S Sukhatme. Automatic differentiation and continuous sensitivity analysis of rigid body dynamics. *arXiv:2001.08539*, 2020.
- [119] Rahul Narain, Armin Samii, and James F. O’Brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph.*, 2012.
- [120] A Witkin and David Baraff. Physically based modeling: Principles and practice. SIGGRAPH Course Notes, 1997.
- [121] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019.
- [122] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IROS*, 2012.
- [123] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *arXiv:1604.00772*, 2016.
- [124] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.
- [125] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH Courses*, 2015.
- [126] Jeongseok Lee, Michael Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha Srinivasa, Mike Stilman, and C. Liu. DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3, 2018.
- [127] Russ Tedrake et al. Drake: Model-based design and verification for robotics. <https://drake.mit.edu>, 2019.
- [128] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.
- [129] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *OSDI*, 2016.
- [130] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2007.
- [131] Michael B Giles and Niles A Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, 65(3-4):393–415, 2000.
- [132] Roy Featherstone. Spatial vector and dynamics software. <http://royfeatherstone.org/spatial/>, 2008.

- [133] Andreas Griewank and Andrea Walther. Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.*, 26(1):19–45, 2000.
- [134] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6), 2018.
- [135] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6), 2020.
- [136] Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39(6), 2020.
- [137] Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. Differentiable programming for image processing and deep learning in Halide. *ACM Transactions on Graphics (TOG)*, 37(4), 2018.
- [138] Jatavallabhula Krishna Murthy, Ganesh Iyer, and Liam Paull. gradSLAM: Dense SLAM meets automatic differentiation. In *ICRA*, 2020.
- [139] Tao Du, Kui Wu, Andrew Spielberg, Wojciech Matusik, Bo Zhu, and Eftychios Sifakis. Functional optimization of fluidic devices with differentiable stokes flow. *ACM Transactions on Graphics (TOG)*, 2020.
- [140] Jakub Stepien. *Physics-Based Animation of Articulated Rigid Body Systems for Virtual Environments*. PhD thesis, Silesian University of Technology, 2013.
- [141] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Neural Information Processing Systems*, 2019.
- [142] Allan Leal et al. autodiff: automatic differentiation made easier for C++. <https://github.com/autodiff/autodiff>, 2018.
- [143] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.
- [144] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *ICML*, 2017.
- [145] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.
- [146] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *ICML*, 2018.

- [147] 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.
- [148] Guorui Li, Xiangping Chen, Fanghao Zhou, Yiming Liang, Youhua Xiao, Xunuo Cao, Zhen Zhang, Mingqi Zhang, Baosheng Wu, Shunyu Yin, et al. Self-powered soft robot in the mariana trench. *Nature*, 591(7848):66–71, 2021.
- [149] Joshua Fishman, Samuel Ubellacker, Nathan Hughes, and Luca Carlone. Dynamic grasping with a “soft” drone: From theory to practice. *arXiv:2103.06465*, 2021.
- [150] Raphael Deimel and Oliver Brock. A compliant hand based on a novel pneumatic actuator. In *ICRA*, 2013.
- [151] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. DiffPD: Differentiable projective dynamics with contact. *arXiv:2101.05917*, 2021.
- [152] Shunichi Kurumaya, Koichi Suzumori, Hiroyuki Nabae, and Shuichi Wakimoto. Musculoskeletal lower-limb robot driven by multifilament muscles. *Robomech Journal*, 3(1):1–15, 2016.
- [153] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *ICRA*, 2018.
- [154] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Joint optimization of robot design and motion parameters using the implicit function theorem. In *Robotics: Science and Systems (RSS)*, 2017.
- [155] Keenon Werling, Dalton Omens, Jeongseok Lee, Ionnis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. In *Robotics: Science and Systems (RSS)*, 2021.
- [156] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Efficient differentiable simulation of articulated bodies. In *ICML*, 2021.
- [157] Kiwon Um, Robert Brand, Yun (Raymond) Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. In *Neural Information Processing Systems*, 2020.
- [158] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S. Sukhatme. NeuralSim: Augmenting differentiable simulators with neural networks. *arXiv:2011.04217*, 2020.
- [159] Xuan Li, Jessica McWilliams, Minchen Li, Cynthia Sung, and Chenfanfu Jiang. Soft hybrid aerial vehicle via bistable mechanism. *arXiv:2011.00426*, 2020.

- [160] 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 Transactions on Graphics (TOG)*, 40(4):132, 2021.
- [161] 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*, 2021.
- [162] Han Shao, Tassilo Kugelstadt, Torsten Hädrich, Wojciech Pałubicki, Jan Bender, Sören Pirk, and Dominik L Michels. Accurately solving physical systems with graph learning. *arXiv preprint arXiv:2006.03897*, 2020.
- [163] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.
- [164] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 — seamless operability between c++11 and python, 2016. <https://github.com/pybind/pybind11>.
- [165] DoodleNotes. Baymax walk cycle animation. <https://blendswap.com/blend/18773>, 2022.
- [166] Mark A Coletti, Eric O Scott, and Jeffrey K Bassett. Library for evolutionary algorithms in Python (LEAP). In *Genetic and Evolutionary Computation Conference*, 2020.
- [167] Michael JD Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06*, University of Cambridge, Cambridge, pages 26–46, 2009.
- [168] Saša Singer and John Nelder. Nelder-Mead algorithm. *Scholarpedia*, 4(7):2928, 2009.
- [169] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.
- [170] Edward Lieberman and Ajay K Rathi. Traffic simulation. *Traffic flow theory*, 1997.
- [171] Shivam Akhauri, Laura Y Zheng, and Ming C Lin. Enhanced transfer learning for autonomous driving with systematic accident simulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5986–5993. IEEE, 2020.
- [172] Jason Sewall, David Wilkie, and Ming C Lin. Interactive hybrid simulation of large-scale traffic. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pages 1–12, 2011.
- [173] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- [174] AATM Aw and Michel Rascle. Resurrection of” second order” models of traffic flow. *SIAM journal on applied mathematics*, 60(3):916–938, 2000.

- [175] H Michael Zhang. A non-equilibrium traffic model devoid of gas-like behavior. *Transportation Research Part B: Methodological*, 36(3):275–290, 2002.
- [176] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.
- [177] Sanghyun Son, Yi-Ling Qiao, Jason Sewall, and Ming C. Lin. Differentiable hybrid traffic simulation. *ACM Transactions on Graphics (TOG)*, 2022.
- [178] Filipe de A. Belbute-Peres, Kevin A. Smith, Kelsey R. Allen, Joshua B. Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, page 7178–7189, 2018.
- [179] Jonas Degraeve, Michiel Hermans, J. Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13, 11 2016.
- [180] Shinjiro Sueda. Analytically differentiable articulated rigid body dynamics, 2020. URL <https://github.com/sueda/redmax/blob/master/notes.pdf>.
- [181] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable programming for physical simulation. *ICLR*, 2020.
- [182] Philipp Holl, Nils Thuerey, and Vladlen Koltun. Learning to control pdes with differentiable physics. In *International Conference on Learning Representations*, 2020.
- [183] Kiwon Um, Raymond, Fei, Philipp Holl, Robert Brand, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers, 2020.
- [184] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020.
- [185] S. Wiewel, M. Becher, and N. Thuerey. Latent space physics: Towards learning the temporal evolution of fluid flow. *Computer Graphics Forum*, 38(2):71–82, 2019.
- [186] Steffen Wiewel, Byungsoo Kim, Vinicius C. Azevedo, Barbara Solenthaler, and Nils Thuerey. Latent space subdivision: Stable and controllable time predictions for fluid flow. In *Symposium on Computer Animation*, 2020.
- [187] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen,

- Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.
- [188] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020. ISSN 0036-8075. doi: 10.1126/science.abe8770. URL <https://science.sciencemag.org/content/370/6523/1460>.
- [189] Y. Shi, P. Gokhale, P. Murali, J. M. Baker, C. Duckering, Y. Ding, N. C. Brown, C. Chamberland, A. Javadi-Abhari, A. W. Cross, D. I. Schuster, K. R. Brown, M. Martonosi, and F. T. Chong. Resource-efficient quantum computing by breaking abstractions. *Proceedings of the IEEE*, 108(8):1353–1370, 2020. doi: 10.1109/JPROC.2020.2994765.
- [190] Alicia B. Magann, Christian Arenz, Matthew D. Grace, Tak-San Ho, Robert L. Kosut, Jarrod R. McClean, Herschel A. Rabitz, and Mohan Sarovar. From pulses to circuits and back again: A quantum optimal control perspective on variational quantum algorithms. *PRX Quantum*, 2:010101, Jan 2021. doi: 10.1103/PRXQuantum.2.010101. URL <https://link.aps.org/doi/10.1103/PRXQuantum.2.010101>.
- [191] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.
- [192] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O’Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):1–7, 2014.
- [193] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.
- [194] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4): 043001, 2019.
- [195] Aram W Harrow and Ashley Montanaro. Quantum computational supremacy. *Nature*, 549 (7671):203–209, 2017.
- [196] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018.

- [197] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh A. Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 2019.
- [198] Shaopeng Zhu, Shih-Han Hung, Shouvanik Chakrabarti, and Xiaodi Wu. On the principles of differentiable quantum programming languages. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2020*, page 272–285, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450376136. doi: 10.1145/3385412.3386011. URL <https://doi.org/10.1145/3385412.3386011>.
- [199] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285. PMLR, 2018.
- [200] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [201] William K Wootters and Wojciech H Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.
- [202] Jiaqi Leng, Yuxiang Peng, Yi-Ling Qiao, Ming C. Lin, and Xiaodi Wu. Differentiable analog quantum computing for optimization and control. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [203] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.
- [204] Kaichun Mo, Shilin Zhu, Angel Chang, Li Yi, Subarna Tripathi, Leonidas Guibas, and Hao Su. PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [205] Atsuhiko Noguchi, Umar Iqbal, Jonathan Tremblay, Tatsuya Harada, and Orazio Gallo. Watch it move: Unsupervised discovery of 3d joints for re-posing of articulated objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3677–3687, 2022.
- [206] Gengshan Yang, Minh Vo, Natalia Neverova, Deva Ramanan, Andrea Vedaldi, and Hanbyul Joo. Banmo: Building animatable 3d neural models from many casual videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2863–2873, 2022.
- [207] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. Neursim: Augmenting differentiable simulators with neural networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9474–9481. IEEE, 2021.

- [208] Xuan Li, Jessica McWilliams, Minchen Li, Cynthia Sung, and Chenfanfu Jiang. Soft hybrid aerial vehicle via bistable mechanism. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7107–7113. IEEE, 2021.
- [209] Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2021.
- [210] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2022. URL <http://github.com/google/brax>.
- [211] Pingchuan Ma, Tao Du, Joshua B. Tenenbaum, Wojciech Matusik, and Chuang Gan. RISP: Rendering-invariant state predictor with differentiable simulation and rendering for cross-domain parameter estimation. In *International Conference on Learning Representations*, 2022.
- [212] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Differentiable simulation of soft multi-body systems. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- [213] Sizhe Li, Zhiao Huang, Tao Du, Hao Su, Joshua Tenenbaum, and Chuang Gan. Contact Points Discovery for Soft-Body Manipulations with Differentiable Physics. In *International Conference on Learning Representations (ICLR)*, 2022.
- [214] Eric Heiden, Miles Macklin, Yashraj S Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. DiSEct: A Differentiable Simulation Engine for Autonomous Robotic Cutting. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.067.
- [215] Yifei Li, Tao Du, Kui Wu, Jie Xu, and Wojciech Matusik. Diffcloth: Differentiable cloth simulation with dry frictional contact. *ACM Trans. Graph.*, 2022.
- [216] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.
- [217] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [218] Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. Nerf-editing: Geometry editing of neural radiance fields. In *Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [219] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020.
- [220] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

- [221] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.
- [222] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised nerf: Fewer views and faster training for free. *arXiv preprint arXiv:2107.02791*, 2021.
- [223] Fanbo Xiang, Zexiang Xu, Miloš Hašan, Yannick Hold-Geoffroy, Kalyan Sunkavalli, and Hao Su. NeuTex: Neural Texture Mapping for Volumetric Neural Rendering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [224] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4): 102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL <https://doi.org/10.1145/3528223.3530127>.
- [225] Jan Novák. *Efficient Many-Light Rendering of Scenes with Participating Media*. PhD thesis, Karlsruhe Institute of Technology”, May 2014.
- [226] Thomas Müller, Marios Papas, Markus Gross, Wojciech Jarosz, and Jan Novák. Efficient rendering of heterogeneous polydisperse granular media. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 35(6):168:1–168:14, December 2016.
- [227] Stefano Esposito, Daniele Baieri, Stefan Zellmann, André Hinkenjann, and Emanuele Rodolà. Kiloneus: Implicit neural representations with real-time global illumination. *arXiv preprint arXiv:2206.10885*, 2022.
- [228] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Mueller, and Sanja Fidler. Extracting Triangular 3D Models, Materials, and Lighting From Images. *arXiv:2111.12503*, 2021.
- [229] Matthew Tancik*, Ethan Weber*, Evonne Ng*, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. Nerfstudio: A framework for neural radiance field development, 2022. URL <https://github.com/nerfstudio-project/nerfstudio>.
- [230] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022.
- [231] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. Xpbd: Position-based simulation of compliant constrained dynamics. In *MIG ’16*, page 49–54. Association for Computing Machinery, 2016. doi: 10.1145/2994258.2994272.
- [232] Jan Novák, Iliyan Georgiev, Johannes Hanika, and Wojciech Jarosz. Monte Carlo methods for volumetric light transport simulation. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)*, 37(2), May 2018.

- [233] Jan Novák, Iliyan Georgiev, Johannes Hanika, Jaroslav Křivánek, and Wojciech Jarosz. Monte carlo methods for physically based volume rendering. In *ACM SIGGRAPH 2018 Courses*, SIGGRAPH '18, pages 14:1–14:1, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5809-5. doi: 10.1145/3214834.3214880. URL <http://doi.acm.org/10.1145/3214834.3214880>.
- [234] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.
- [235] Thomas Müller, Marios Papas, Markus Gross, Wojciech Jarosz, and Jan Novák. Efficient rendering of heterogenous polydisperse granular media. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2016)*, 35(6):168:1–168:14, 2016. doi: 10.1145/2980179.2982429.
- [236] Binh-Son Hua, Adrien Gruson, Victor Petitjean, Matthias Zwicker, Derek Nowrouzezahrai, Elmar Eisemann, and Toshiya Hachisuka. A survey on gradient-domain rendering. In *Computer Graphics Forum*, volume 38, pages 455–472. Wiley Online Library, 2019.
- [237] Bailey Miller, Iliyan Georgiev, and Wojciech Jarosz. A null-scattering path integral formulation of light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 38(4), July 2019. ISSN 0730-0301. doi: 10/gf6rzb.
- [238] Merlin Nimier-David, Thomas Müller, Alexander Keller, and Wenzel Jakob. Unbiased inverse volume rendering with differential trackers. *ACM Trans. Graph.*, 2022.
- [239] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *arXiv*, 2021.
- [240] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. *arXiv preprint arXiv:2201.08845*, 2022.
- [241] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020.
- [242] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022.
- [243] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022.
- [244] Matthew Tancik, Vincent Casser, Xinchun Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8248–8258, 2022.

- [245] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In *CVPR*, 2022.
- [246] Konstantinos Rematas, Andrew Liu, Pratul P Srinivasan, Jonathan T Barron, Andrea Tagliasacchi, Thomas Funkhouser, and Vittorio Ferrari. Urban radiance fields. In *CVPR*, 2022.
- [247] Georgios Kopanas, Thomas Leimkühler, Gilles Rainer, Clément Jambon, and George Drettakis. Neural point catacaustics for novel-view synthesis of reflections. *ACM Transactions on Graphics*, 41(6):Article–201, 2022.
- [248] Ziyu Wang, Liao Wang, Fuqiang Zhao, Minye Wu, Lan Xu, and Jingyi Yu. Mirrornerf: One-shot neural portrait radiance field from multi-mirror catadioptric imaging. In *2021 IEEE International Conference on Computational Photography (ICCP)*, pages 1–12. IEEE, 2021.
- [249] Yu-Lun Liu, Chen Gao, Andreas Meuleman, Hung-Yu Tseng, Ayush Saraf, Changil Kim, Yung-Yu Chuang, Johannes Kopf, and Jia-Bin Huang. Robust dynamic radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.
- [250] Luma AI. Luma ai. <https://lumalabs.ai/>, 2022.
- [251] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *ICCV*, 2021.
- [252] Huan Wang, Jian Ren, Zeng Huang, Kyle Olszewski, Menglei Chai, Yun Fu, and Sergey Tulyakov. R2l: Distilling neural radiance field to neural light field for efficient novel view synthesis. *arXiv preprint arXiv:2203.17261*, 2022.
- [253] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv preprint arXiv:2208.00277*, 2022.
- [254] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *ICCV*, 2021.
- [255] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. Real-time neural radiance caching for path tracing. *ACM Trans. Graph.*, 40(4):36:1–36:16, August 2021. doi: 10.1145/3450626.3459812. URL <https://doi.org/10.1145/3450626.3459812>.
- [256] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021.
- [257] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenotrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021.

- [258] Benjamin Attal, Jia-Bin Huang, Michael Zollhöfer, Johannes Kopf, and Changil Kim. Learning neural light fields with ray-space embedding. In *CVPR*, 2022.
- [259] Benjamin Attal, Jia-Bin Huang, Christian Richardt, Michael Zollhoefer, Johannes Kopf, Matthew O’Toole, and Changil Kim. Hyperreel: High-fidelity 6-dof video with ray-conditioned sampling. In *CVPR*, 2023.
- [260] Liao Wang, Jiakai Zhang, Xinhang Liu, Fuqiang Zhao, Yanshun Zhang, Yingliang Zhang, Minye Wu, Jingyi Yu, and Lan Xu. Fourier plenotrees for dynamic radiance field rendering in real-time. In *CVPR*, 2022.
- [261] Yoni Kasten, Dolev Ofri, Oliver Wang, and Tali Dekel. Layered neural atlases for consistent video editing. *ACM Transactions on Graphics (TOG)*, 40(6):1–12, 2021.
- [262] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *CVPR*, 2021.
- [263] Viktor Rudnev, Mohamed Elgharib, William Smith, Lingjie Liu, Vladislav Golyanik, and Christian Theobalt. Neural radiance fields for outdoor scene relighting. *arXiv preprint arXiv:2112.05140*, 2021.
- [264] Pratul P Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *CVPR*, 2021.
- [265] Bangbang Yang, Yinda Zhang, Yinghao Xu, Yijin Li, Han Zhou, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. Learning object-compositional neural radiance field for editable scene rendering. In *ICCV*, 2021.
- [266] Zian Wang, Tianchang Shen, Jun Gao, Shengyu Huang, Jacob Munkberg, Jon Hasselgren, Zan Gojcic, Wenzheng Chen, and Sanja Fidler. Neural fields meet explicit geometric representations for inverse rendering of urban scenes. In *CVPR*, 2023.
- [267] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, Light, and Material Decomposition from Images using Monte Carlo Rendering and Denoising. *arXiv:2206.03380*, 2022.
- [268] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL <http://www.blender.org>.
- [269] Yi-Ling Qiao, Alexander Gao, and Ming C. Lin. Neuphysics: Editable neural geometry and physics from monocular videos. In *Neural Information Processing Systems*, 2022.
- [270] Mengyu Chu, Lingjie Liu, Quan Zheng, Erik Franz, Hans-Peter Seidel, Christian Theobalt, and Rhaleb Zayer. Physics informed neural fields for smoke reconstruction with sparse data. *ACM Transactions on Graphics*, 2022.

- [271] Simon Cleac’h, Hong-Xing Yu, Michelle Guo, Taylor Howell, Ruohan Gao, Jiajun Wu, Zac Manchester, and Mac Schwager. Differentiable physics simulation of dynamics-augmented neural objects. *IEEE Robotics and Automation Letters*, 10 2022.
- [272] Xuan Li, Yi-Ling Qiao, Peter Yichen Chen, Krishna Murthy Jatavallabhula, Ming Lin, Chenfanfu Jiang, and Chuang Gan. Pac-nerf: Physics augmented continuum neural radiance fields for geometry-agnostic system identification. *ICLR*, 2023.
- [273] Jianchuan Chen, Ying Zhang, Di Kang, Xuefei Zhe, Linchao Bao, Xu Jia, and Huchuan Lu. Animatable neural radiance fields from monocular rgb videos, 2021.
- [274] Sida Peng, Yuanqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans. In *CVPR*, 2021.
- [275] Chung-Yi Weng, Brian Curless, Pratul P. Srinivasan, Jonathan T. Barron, and Ira Kemelmacher-Shlizerman. HumanNeRF: Free-viewpoint rendering of moving people from monocular video. In *CVPR*, pages 16210–16220, June 2022.
- [276] Lingjie Liu, Marc Habermann, Viktor Rudnev, Kripasindhu Sarkar, Jiatao Gu, and Christian Theobalt. Neural actor: Neural free-view synthesis of human actors with pose control. *ACM Trans. Graph.(ACM SIGGRAPH Asia)*, 2021.
- [277] Michelle Guo, Alireza Fathi, Jiajun Wu, and Thomas Funkhouser. Object-centric neural scene rendering, 2022. URL <https://openreview.net/forum?id=Uy6YEI9-6v>.
- [278] Yuan Li, Zhi-Hao Lin, David Forsyth, Jia-Bin Huang, and Shenlong Wang. Climatenerf: Physically-based neural rendering for extreme climate synthesis. *arXiv e-prints*, pages arXiv-2211, 2022.
- [279] Yao Yao, Jingyang Zhang, Jingbo Liu, Yihang Qu, Tian Fang, David McKinnon, Yanghai Tsin, and Long Quan. Neilf: Neural incident light field for physically-based material estimation. In *ECCV*, page 700–716, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-19820-5. doi: 10.1007/978-3-031-19821-2_40.
- [280] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL <https://arxiv.org/pdf/2205.15585.pdf>.
- [281] Kevin Karsch, Varsha Hedau, David Forsyth, and Derek Hoiem. Rendering synthetic objects into legacy photographs. *ACM Trans. Graph.*, 30(6):1–12, dec 2011. ISSN 0730-0301. doi: 10.1145/2070781.2024191. URL <https://doi.org/10.1145/2070781.2024191>.
- [282] Yun Chen, Frieda Rong, Shivam Duggal, Shenlong Wang, Xinchun Yan, Sivabalan Manivasagam, Shangjie Xue, Ersin Yumer, and Raquel Urtasun. Geosim: Realistic video simulation via geometry-aware composition for self-driving. In *CVPR*, 2021.

- [283] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P. Srinivasan, and Jonathan T. Barron. NeRF in the dark: High dynamic range view synthesis from noisy raw images. *CVPR*, 2022.
- [284] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [285] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. Dr.jit: A just-in-time compiler for differentiable rendering. *Transactions on Graphics (Proceedings of SIGGRAPH)*, 41(4), July 2022. doi: 10.1145/3528223.3530099.
- [286] Merlin Nimier-David, Zhao Dong, Wenzel Jakob, and Anton Kaplanyan. Material and Lighting Reconstruction for Complex Indoor Scenes with Texture-space Differentiable Rendering. In Adrien Bousseau and Morgan McGuire, editors, *Eurographics Symposium on Rendering - DL-only Track*. The Eurographics Association, 2021. ISBN 978-3-03868-157-1. doi: 10.2312/sr.20211292.
- [287] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022.
- [288] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. Xpbd: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, pages 49–54, 2016.
- [289] Matthias Müller, Miles Macklin, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim. Detailed rigid body simulation with extended position based dynamics. In *Computer Graphics Forum*, volume 39, pages 101–112. Wiley Online Library, 2020.
- [290] Jiaxiang Tang, Xiaokang Chen, Jingbo Wang, and Gang Zeng. Compressible-composable nerf via rank-residual decomposition. *arXiv preprint arXiv:2205.14870*, 2022.
- [291] Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. <https://github.com/nvidia/warp>, March 2022. NVIDIA GPU Technology Conference (GTC).
- [292] Paul Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH*, 1998.
- [293] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. *SIGGRAPH 97*, 1997.
- [294] baehappy1322. The all new ford bronco, 2022. URL <https://skfb.ly/oCMKr>.
- [295] toivo. Car scene, 2022. URL <https://skfb.ly/6uSoM>.
- [296] Ivo. High-poly modern wood chair, 2022. URL <https://skfb.ly/6THKA>.

- [297] robula72. coffee table marina, 2022. URL <https://skfb.ly/6WM8x>.
- [298] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, October 2015.
- [299] Guillermo Garcia-Hernando, Shanxin Yuan, Seungryul Baek, and Tae-Kyun Kim. First-person hand action benchmark with rgb-d videos and 3d hand pose annotations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 409–419, 2018.
- [300] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.
- [301] Ram Pratap Sharma and Gyanendra K Verma. Human computer interaction using hand gesture. *Procedia Computer Science*, 54:721–727, 2015.
- [302] Yana Hasson, Gül Varol, Dimitris Tzionas, Igor Kalevatykh, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning joint reconstruction of hands and manipulated objects. In *CVPR*, 2019.
- [303] Shaowei Liu, Hanwen Jiang, Jiarui Xu, Sifei Liu, and Xiaolong Wang. Semi-supervised 3d hand-object poses estimation with interactions in time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2021.
- [304] Zhe Cao, Ilija Radosavovic, Angjoo Kanazawa, and Jitendra Malik. Reconstructing hand-object interactions in the wild. *ICCV*, 2021.
- [305] Y. Hasson, G. Varol, C. Schmid, and I. Laptev. Towards unconstrained joint hand-object reconstruction from rgb videos. In *2021 International Conference on 3D Vision (3DV)*, 2021.
- [306] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.
- [307] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [308] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43(46):3736–3741, 2004.
- [309] Shreyas Hampali, Mahdi Rad, Markus Oberweger, and Vincent Lepetit. Honnotate: A method for 3d annotation of hand and object poses. In *CVPR*, 2020.

- [310] Shreyas Hampali, Sayan Deb Sarkar, Mahdi Rad, and Vincent Lepetit. Keypoint transformer: Solving joint identification in challenging hands and object interactions for accurate 3d pose estimation. In *CVPR*, 2022.
- [311] Gideon Billings and Matthew Johnson-Roberson. Silhonet: An rgb method for 6d object pose estimation. *IEEE Robotics and Automation Letters*, 4(4):3727–3734, 2019.
- [312] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 536–551, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10605-2.
- [313] Yinlin Hu, Joachim Hugonot, Pascal Fua, and Mathieu Salzmann. Segmentation-driven 6d object pose estimation. In *CVPR*, 2019.
- [314] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-Time Seamless Single Shot 6D Object Pose Prediction. In *CVPR*, 2018.
- [315] Yisheng He, Haibin Huang, Haoqiang Fan, Qifeng Chen, and Jian Sun. Ffb6d: A full flow bidirectional fusion network for 6d pose estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021.
- [316] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. In *Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [317] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*, 2018.
- [318] Javier Romero, Dimitrios Tzionas, and Michael J. Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 2017.
- [319] Yujun Cai, Liuhao Ge, Jianfei Cai, and Junsong Yuan. Weakly-supervised 3d hand pose estimation from monocular rgb images. In *European Conference on Computer Vision*, 2018.
- [320] Liuhao Ge, Yujun Cai, Junwu Weng, and Junsong Yuan. Hand pointnet: 3d hand pose estimation using point sets. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.
- [321] Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, and Christian Theobalt. Gnerated hands for real-time 3d hand tracking from monocular rgb. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2018.

- [322] Adrian Spurr, Jie Song, Seonwook Park, and Otmar Hilliges. Cross-modal deep variational hand pose estimation. In *CVPR*, 2018.
- [323] Christian Zimmermann and Thomas Brox. Learning to estimate 3d hand pose from single rgb images. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [324] L. Yang and A. Yao. Disentangling latent hands for image synthesis and pose estimation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [325] S. Yuan, G. Garcia-Hernando, B. Stenger, G. Moon, J. Chang, K. Lee, P. Molchanov, J. Kautz, S. Honari, L. Ge, J. Yuan, X. Chen, G. Wang, F. Yang, K. Akiyama, Y. Wu, Q. Wan, M. Madadi, S. Escalera, S. Li, D. Lee, I. Oikonomidis, A. Argyros, and T. Kim. Depth-based 3d hand pose estimation: From current achievements to future goals. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [326] S. Baek, K. Kim, and T. Kim. Pushing the envelope for rgb-based dense 3d hand pose estimation via neural rendering. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [327] Adnane Boukhayma, Rodrigo de Bem, and Philip HS Torr. 3d hand shape and pose from images in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10843–10852, 2019.
- [328] Yu Rong, Takaaki Shiratori, and Hanbyul Joo. Frankmocap: A monocular 3d whole-body pose estimation system via regression and integration. In *IEEE International Conference on Computer Vision Workshops*, 2021.
- [329] Christian Zimmermann, Duygu Ceylan, Jimei Yang, Bryan Russel, Max Argus, and Thomas Brox. Freihand: A dataset for markerless capture of hand pose and shape from single rgb images. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. URL ["https://lmb.informatik.uni-freiburg.de/projects/freihand/"](https://lmb.informatik.uni-freiburg.de/projects/freihand/).
- [330] Dominik Kulon, Riza Alp Guler, Iasonas Kokkinos, Michael M. Bronstein, and Stefanos Zafeiriou. Weakly-supervised mesh-convolutional hand reconstruction in the wild. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [331] Gyeongsik Moon, Shoou-I Yu, He Wen, Takaaki Shiratori, and Kyoung Mu Lee. Interhand2.6m: A dataset and baseline for 3d interacting hand pose estimation from a single rgb image. In *European Conference on Computer Vision (ECCV)*, 2020.
- [332] Franziska Mueller, Dushyant Mehta, Oleksandr Sotnychenko, Srinath Sridhar, Dan Casas, and Christian Theobalt. Real-time hand tracking under occlusion from an egocentric rgb-d sensor. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2017. URL <http://handtracker.mpi-inf.mpg.de/projects/OccludedHands/>.

- [333] P. Panteleris, I. Oikonomidis, and A. Argyros. Using a single rgb frame for real time 3d hand pose estimation in the wild. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018.
- [334] Yunze Liu, Yun Liu, Che Jiang, Kangbo Lyu, Weikang Wan, Hao Shen, Boqiang Liang, Zhoujie Fu, He Wang, and Li Yi. Hoi4d: A 4d egocentric dataset for category-level human-object interaction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21013–21022, June 2022.
- [335] Samarth Brahmabhatt, Chengcheng Tang, Christopher D. Twigg, Charles C. Kemp, and James Hays. ContactPose: A dataset of grasps with object contact and hand pose. In *The European Conference on Computer Vision (ECCV)*, August 2020.
- [336] Yu-Wei Chao, Wei Yang, Yu Xiang, Pavlo Molchanov, Ankur Handa, Jonathan Tremblay, Yashraj S. Narang, Karl Van Wyk, Umar Iqbal, Stan Birchfield, Jan Kautz, and Dieter Fox. DexYCB: A benchmark for capturing hand grasping of objects. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [337] Guillermo Garcia-Hernando, Shanxin Yuan, Seungryul Baek, and Tae-Kyun Kim. First-person hand action benchmark with rgb-d videos and 3d hand pose annotations. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [338] Lixin Yang, Kailin Li, Xinyu Zhan, Jun Lv, Wenqiang Xu, Jiefeng Li, and Cewu Lu. ArtiBoost: Boosting articulated 3d hand-object pose estimation via online exploration and synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [339] JoonKyu Park, Yeonguk Oh, Gyeongsik Moon, Hongsuk Choi, and Kyoung Mu Lee. Handocnet: Occlusion-robust 3d hand mesh estimation network. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [340] Patrick Grady, Chengcheng Tang, Christopher D Twigg, Minh Vo, Samarth Brahmabhatt, and Charles C Kemp. Contactopt: Optimizing contact to improve grasps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1471–1481, 2021.
- [341] Haoyu Hu, Xinyu Yi, Hao Zhang, Jun-Hai Yong, and Feng Xu. Physical interaction: Reconstructing hand-object interactions with physics. In *SIGGRAPH Asia 2022 Conference Papers*, 2022.
- [342] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.
- [343] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An end-to-end differentiable framework for contact-aware robot design. *arXiv preprint arXiv:2107.07501*, 2021.

- [344] Eric Heiden, Christopher E. Denniston, David Millard, Fabio Ramos, and Gaurav S. Sukhatme. Probabilistic inference of simulation parameters via parallel differentiable simulation. *International Conference on Robotics and Automation (ICRA)*, 2022.
- [345] Sammy Christen, Muhammed Kocabas, Emre Aksan, Jemin Hwangbo, Jie Song, and Otmar Hilliges. D-grasp: Physically plausible dynamic grasp synthesis for hand-object interactions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [346] Dylan Turpin, Liquan Wang, Eric Heiden, Yun-Chun Chen, Miles Macklin, Stavros Tsogkas, Sven Dickinson, and Animesh Garg. Grasp’d: Differentiable contact-rich grasp synthesis for multi-fingered hands. In *European Conference on Computer Vision (ECCV)*, 2022.
- [347] Sai Kumar Dwivedi, Nikos Athanasiou, Muhammed Kocabas, and Michael J Black. Learning to regress bodies from images using differentiable semantic rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11250–11259, 2021.
- [348] Jason Y. Zhang, Sam Pepose, Hanbyul Joo, Deva Ramanan, Jitendra Malik, and Angjoo Kanazawa. Perceiving 3d human-object spatial arrangements from a single image in the wild. In *European Conference on Computer Vision (ECCV)*, 2020.
- [349] Erik Gärtner, Mykhaylo Andriluka, Erwin Coumans, and Cristian Sminchisescu. Differentiable dynamics for articulated 3d human motion reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13190–13200, 2022.
- [350] A. Kirillov, Y. Wu, K. He, and R. Girshick. Pointrend: Image segmentation as rendering. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [351] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *European Conference on Computer Vision (ECCV)*, 2014.
- [352] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 International Conference on Advanced Robotics (ICAR)*, 2015.
- [353] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.
- [354] Samarth Brahmabhatt, Chengcheng Tang, Christopher D Twigg, Charles C Kemp, and James Hays. Contactpose: A dataset of grasps with object contact and hand pose. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16*, pages 361–378. Springer, 2020.

- [355] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6), 2020.
- [356] Hang Ma, Sven Koenig, Nora Ayanian, Liron Cohen, Wolfgang Hönig, TK Kumar, Tansel Uras, Hong Xu, Craig Tovey, and Guni Sharon. Overview: Generalizations of multi-agent path finding to real-world scenarios. *arXiv preprint arXiv:1702.05515*, 2017.
- [357] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.
- [358] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.
- [359] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.
- [360] Amichai Levy, Chris Keitel, Sam Engel, and James McLurkin. The extended velocity obstacle and applying orca in the real world. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16–22. IEEE, 2015.
- [361] Mingming Li, Rui Jiang, Shuzhi Sam Ge, and Tong Heng Lee. Role playing learning for socially concomitant mobile robot navigation. *CAAI Transactions on Intelligence Technology*, 3(1):49–58, 2018.
- [362] Adarsh Jagan Sathyamoorthy, Jing Liang, Utsav Patel, Tianrui Guan, Rohan Chandra, and Dinesh Manocha. Denscavoid: Real-time navigation in dense crowds using anticipatory behaviors. *arXiv preprint arXiv:2002.03038*, 2020.
- [363] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots. *arXiv e-prints*, art. arXiv:1609.07910, Sep 2016.
- [364] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1609.05143, Sep 2016.
- [365] B. Gopalakrishnan, A. K. Singh, M. Kaushik, K. M. Krishna, and D. Manocha. Prvo: Probabilistic reciprocal velocity obstacle for multi robot navigation under uncertainty. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1089–1096, Sep. 2017. doi: 10.1109/IROS.2017.8202279.
- [366] Jae Sung Park, Chonhyon Park, and Dinesh Manocha. Efficient probabilistic collision detection for non-convex shapes. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1944–1951. IEEE, 2017.
- [367] Masahiro Shiomi, Francesco Zanlungo, Kotaro Hayashi, and Takayuki Kanda. Towards a socially acceptable collision avoidance for a mobile robot navigating among pedestrians using a pedestrian model. *International Journal of Social Robotics*, 6(3):443–455, 2014.

- [368] Aniket Bera, Tanmay Randhavane, Rohan Prinja, and Dinesh Manocha. Sociosense: Robot navigation amongst pedestrians with social and psychological constraints. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7018–7025. IEEE, 2017.
- [369] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Fully distributed multi-robot collision avoidance via deep reinforcement learning for safe and efficient navigation in complex scenarios. *arXiv preprint arXiv:1808.03841*, 2018.
- [370] Jing Liang, Yi-Ling Qiao, Tianrui Guan, and Dinesh Manocha. Of-vo: efficient navigation among pedestrians using commodity sensors. *IEEE Robotics and Automation Letters*, 6(4):6148–6155, 2021.
- [371] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [372] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, W Yifan, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, et al. Advances in neural rendering. In *Computer Graphics Forum*, volume 41, pages 703–735. Wiley Online Library, 2022.
- [373] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Computer Graphics Forum*, 2022. ISSN 1467-8659. doi: 10.1111/cgf.14505.
- [374] Miguel Jaques, Martin Asenov, Michael Burke, and Timothy Hospedales. Vision-based system identification and 3d keypoint discovery using dynamics constraints. In *Learning for Dynamics and Control Conference*, pages 316–329. PMLR, 2022.
- [375] Miguel Jaques, Michael Burke, and Timothy Hospedales. Physics-as-inverse-graphics: Unsupervised physical parameter estimation from video. *ICLR*, 2020.
- [376] Anthony James Merrill Spencer. *Continuum mechanics*. Courier Corporation, 2004.
- [377] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022.
- [378] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 34(4):1–10, 2015.
- [379] Yi-Ling Qiao, Alexander Gao, Yiran Xu, Yue Feng, Jia-Bin Huang, and Ming C. Lin. Dynamic mesh-aware radiance fields. *ICCV*, 2023.
- [380] Shutong Zhang, Yiling Qiao, Guanglei Zhu, Eric Heiden, Dylan Turpin, Jingzhou Liu, Ming Lin, Miles Macklin, and Animesh Garg. Handypriors: Physically consistent perception of hand-object interactions with differentiable priors. In *ICRA*, 2024.

- [381] Towards generalist robots: Learning paradigms for scalable skill acquisition. <https://generalist-robots.github.io/>, 2023.
- [382] Zhou Xian, Theophile Gervet, Zhenjia Xu, Yi-Ling Qiao, and Tsun-Hsuan Wang. Towards a foundation model for generalist robots: Diverse skill learning at scale via automated task and scene generation. *arXiv preprint arXiv:2305.10455*, 2023.
- [383] Zehao Yu, Anpei Chen, Bozidar Antic, Songyou Peng Peng, Apratim Bhattacharyya, Michael Niemeyer, Siyu Tang, Torsten Sattler, and Andreas Geiger. Sdfstudio: A unified framework for surface reconstruction, 2022. URL <https://github.com/autonomousvision/sdfstudio>.
- [384] Brent Burley. Physically-based shading at disney. In *SIGGRAPH*, 2012.
- [385] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Path replay backpropagation: Differentiating light paths using constant memory and linear time. *Transactions on Graphics (Proceedings of SIGGRAPH)*, 40(4):108:1–108:14, August 2021. doi: 10.1145/3450626.3459804.
- [386] Olga Sorkine-Hornung and Michael Rabinovich. Least-squares rigid motion using svd. *Computing*, 1(1):1–5, 2017.
- [387] Michelle A Lee, Brent Yi, Roberto Martín-Martín, Silvio Savarese, and Jeannette Bohg. Multimodal sensor fusion with differentiable filters. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10444–10451. IEEE, 2020.