

ABSTRACT

Title of Dissertation: Differentiable Systems for Efficient Optimization and Learning:
FROM KINEMATICS TO GEOMETRY

Sanghyun Son
Doctor of Philosophy, 2025

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

Modern scientific and engineering challenges rooted in physics often require searching immense solution spaces. Sampling these high-dimensional spaces can be computationally prohibitive. I observe that many underlying dynamical processes are nearly everywhere differentiable: physical states evolve smoothly in time, and even brief discontinuities preserve overall continuity. When a system's next state and its gradient with respect to the current state are available, those gradients act as beacons that steer optimization far more efficiently than gradient-free exploration. More broadly, gradients capture how changes in one set of parameters induce changes in system outputs, and thus provide a general mechanism for guiding optimization—even in systems that are not strictly continuous. In robotics, for example, differentiable physics allows us to train controllers in very high-dimensional action spaces and to learn policies that operate robustly in contact-rich environments, where gradients propagate through complex interactions between bodies. In 3D reconstruction, differentiable formulations enable efficient optimization over high-dimensional visual cues aggregated from many viewpoints, making it feasible to recover detailed 3D geometry from multi-view observations without

resorting to exhaustive search. Consequently, researchers are investing heavily in differentiable models of real-world phenomena, especially within physical sciences and engineering, where the laws of motion and interaction naturally provide the required smoothness.

Building on these observations across natural and engineered systems, this thesis demonstrates how exploiting differentiability can systematically accelerate problem-solving in both decision making and geometric reconstruction. I first develop gradient-aware reinforcement learning methods that explicitly incorporate system-level derivatives into policy learning: one that augments Proximal Policy Optimization (PPO) with analytic gradients to constrain policy updates and speed convergence, and another that trains a continuous-time world model by randomly sampling time steps, greatly improving sample efficiency. Across these studies, a common insight emerges: exposing derivatives of a system’s dynamics turns exploration in high-dimensional control spaces into a far more efficient and directed optimization process. This perspective extends beyond continuous-time dynamics: even in systems with discrete structures or topological changes, useful gradients can be obtained by relaxing, approximating, or reparameterizing the underlying representations. Motivated by this insight, I then focus on implementing and strengthening differentiability in complex physical systems themselves, so that their internal gradients can be directly leveraged for downstream tasks. The first system is a GPU-accelerated traffic simulator with exact vehicle-kinematic derivatives, enabling rapid optimization of autonomous vehicle policies and traffic-signal timing in road networks. The second is a family of differentiable mesh formulations that make traditionally discrete triangle and tetrahedral meshes amenable to gradient-based optimization, including a probabilistic triangular-mesh representation that relaxes edge existence into continuous variables and a differentiable tetrahedral-mesh extraction pipeline for 3D object and mesh reconstruction. Although mesh connectivity and topology are inherently discrete, these methods reformulate geometric relationships so that gradients—interpreted

as sensitivities between geometric parameters and observations—can be computed and exploited for efficient optimization. By weaving these case studies together, the thesis articulates a unified path toward simulators and learned agents that use physics-based gradients to reason more effectively about the real world, support scalable optimization for robotics and autonomous systems, and enable high-fidelity, differentiable 3D representations of our environment.

Differentiable Systems for Efficient Optimization and Learning:
From Kinematics to Geometry

by

Sanghyun Son

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
2025

Advisory Committee:

Professor Ming C. Lin, Chair/Advisor
Professor Haizhao Yang, Dean's Representative
Professor Dinesh Manocha
Professor Ruohan Gao
Dr. Yi Zhou

© Copyright by
Sanghyun Son
2025

Acknowledgments

I would like to express my deepest gratitude to my advisor, Professor Ming C. Lin, for her guidance, support, and trust throughout my Ph.D. journey. Under her mentorship, I learned how to develop differentiable systems that integrate geometry, physics simulation, and optimization, and how to translate fundamental ideas into scalable and practical research contributions. Her perspective on research has profoundly shaped the way I approach problem formulation, system design, and long-term impact. I am especially grateful for the intellectual freedom she provided, along with her steady encouragement and thoughtful guidance at key moments throughout my doctoral studies.

I am also deeply thankful to Professor Myung-Soo Kim, my master’s advisor, for his guidance during my master’s studies. Under his mentorship, I built a strong foundation in computer graphics, geometry, and physics simulation, which later became the basis for my doctoral research. His early guidance and encouragement played an important role in shaping my path toward pursuing a Ph.D.

I would like to thank my dissertation committee members—Professor Haizhao Yang, Professor Dinesh Manocha, Professor Ruohan Gao, and Dr. Yi Zhou—for their time, careful reading, and valuable feedback. Their questions and comments helped improve the clarity, rigor, and scope of this thesis, and their diverse perspectives contributed meaningfully to refining the ideas presented in this work.

Beyond my committee, I am grateful to my collaborators for their support, discussions, and shared efforts throughout my Ph.D. I would like to thank Seunghyun Yoon, Gershon Elber, Youngjin Park, Yi-Ling Qiao, Jason Sewall, Laura Yu Zheng, Ryan Sullivan, Peng Gao, Jing Liang, Yu Shen, Xijun Wang, Brian Clipp, Matheus Gadelha, Yang Zhou, Zexiang Xu, Connor Greenwell, Sujin Philip,

Matthew Fisher, Anh N. Nhu, Jan Held, Renaud Vandeghen, Daniel Rebain, Marc Van Droogenbroeck, and Andrea Tagliasacchi for many valuable discussions and collaborations that contributed to this work.

Among them, I am especially grateful to Dr. Yi Zhou for her constant support and encouragement during my internship at Adobe Research and throughout my work on differentiable mesh representations. Her trust, insight, and unwavering support were a significant source of motivation and had a lasting impact on my research. I would also like to thank Yi-Ling Qiao, my lab mate, who played an important role in helping me adapt when I first joined the lab. His approach to research, along with his generosity in sharing advice and experience, provided a strong example of how to conduct thoughtful and rigorous research during my Ph.D.

I am also grateful to my lab mates for creating a supportive and stimulating research environment throughout my doctoral studies. I would like to thank Yi-Ling Qiao, Laura Yu Zheng, Jing Liang, Xijun Wang, Alexander Gao, Jaehoon Choi, Dongki Jung, Shrey Patel, Tianrui Guan, Yonghan Lee, Anh Nhu, and Peng Gao for many discussions, collaborations, and everyday interactions that made the research process both productive and enjoyable. Working alongside such dedicated and thoughtful colleagues greatly enriched my experience during my Ph.D.

I would also like to acknowledge the institutional and administrative support that made this work possible. At the University of Maryland, I am grateful to Tom, Jodie, Sharron, and Migo for their assistance and support throughout my doctoral studies. Their help with administrative and logistical matters was invaluable and allowed me to focus on my research. I am also thankful to Adobe Research for providing me with two internship opportunities and a supportive research environment, which played an important role in shaping my research direction and broadening my perspective on industry-driven research. Finally, I would like to thank Genesis AI for the opportunity to intern and engage with

applied research in a dynamic and collaborative setting.

Finally, I would like to express my deepest gratitude to my parents and my sister for their unwavering support and encouragement. Without their belief in me, this journey would not have been possible. I am especially thankful to my closest friend and partner, Geonsun Lee, whose support and understanding provided a steady foundation for both my research and daily life, and whose presence was essential in helping me complete my Ph.D. with confidence and stability. I would also like to extend my sincere thanks to one of my closest friends, Jongho Lee, for his constant encouragement and support. In addition, I am grateful to all of my friends, whose friendship and support—near and far—meant more to me than words can express. Without all of this support, completing my doctoral studies would not have been possible.

Table of Contents

Acknowledgements	ii
Table of Contents	v
List of Tables	xi
List of Figures	xiii

I Introduction 1

Chapter 1: Overview	2
1.1 Motivation	2
1.2 Technical Challenges	5
1.3 Problem Statements	7
1.4 Thesis Statement and Hypotheses	10
1.5 Overview of Thematic Approaches	12
1.6 Related Work	15
1.6.1 Reinforcement Learning for Physical Tasks	15
1.6.2 Differentiable Kinematics	17
1.6.3 Differentiable Geometry	20
1.7 Outline and Contributions	21

II Reinforcement Learning for Physical Tasks 26

Chapter 2: Gradient Informed Proximal Policy Optimization	27
2.1 Introduction	27
2.2 Related Work	29
2.2.1 Policy Learning Without Analytical Gradients	29
2.2.2 Policy Learning With Analytical Gradients	30
2.3 Preliminaries	31
2.3.1 Goal	31
2.3.2 Analytical Gradients	31
2.3.3 Policy Update	32
2.4 Approach	34
2.4.1 α -Policy	34
2.4.2 Relationship between RP gradient and α -policy	36
2.4.3 Algorithm	37
2.5 Experimental Results	40
2.5.1 Function Optimization Problems	42
2.5.2 Differentiable Physics Simulation	44

2.5.3	Traffic Problems	45
2.6	Conclusions	46
Chapter 3:	Time-Aware World Model for Adaptive Prediction and Control	49
3.1	Introduction	49
3.2	Related Work	53
3.3	Background and Motivations	54
3.3.1	Model-Based Reinforcement Learning	54
3.3.2	Theoretical Motivations	55
3.4	Methodology	57
3.4.1	Model Architecture	58
3.4.2	Training Procedure Using a Mixture of Δt	61
3.4.3	Theoretical Analysis of TAWM's Sample Efficiency	62
3.5	Experiments	64
3.5.1	Evaluations on Control Problems	64
3.5.2	Ablation studies on sampling strategies	70
3.6	Conclusion	71
III	Differentiable Kinematics: Traffic	73
Chapter 4:	Differentiable Hybrid Traffic Simulation	74
4.1	Introduction	74
4.2	Previous Work	77
4.2.1	Traffic models	77
4.2.2	Differentiable models	78
4.3	Macroscopic Model	79
4.3.1	Formulation	79
4.3.2	Numerical Solution	80
4.3.3	Differentiation	81
4.4	Microscopic Model	82
4.4.1	Formulation	82
4.4.2	Differentiation	83
4.5	Macro-Micro-Macro Conversion	84
4.5.1	Macro to Micro	84
4.5.2	Micro to Macro	85
4.5.3	Differentiation	86
4.6	Experimental Results	87
4.6.1	Acceleration	88
4.6.2	Parameter Estimation Problem	90
4.6.3	Traffic Control Problems	92
4.7	Conclusions	97
4.7.1	Generalization	98
4.7.2	Limitations and Future Directions	99

Chapter 5:	Gradient-based Trajectory Optimization with Parallelized Differentiable Traffic Simulation	101
5.1	Introduction	101
5.2	Related Work	104
5.2.1	Traffic Simulators	104
5.2.2	Trajectory Optimization	105
5.3	Method	106
5.3.1	Preliminary	106
5.3.2	Parallelization Scheme	108
5.3.3	Unrealistic Behaviors	108
5.4	Application	109
5.4.1	Trajectory Filtering & Reconstruction	109
5.4.2	Trajectory Prediction	110
5.5	Experimental Results	111
5.5.1	Computational Efficiency	112
5.5.2	Trajectory Filtering & Reconstruction	113
5.5.3	A Training-Free Trajectory Forecasting Baseline	117
5.6	Conclusion	119

IV Differentiable Geometry: Mesh 121

Chapter 6:	DMesh: A Differentiable Mesh Representation	122
6.1	Introduction	123
6.2	Related Work	125
6.2.1	Shape Representations for Optimization	125
6.2.2	Delaunay Triangulation for Geometry Processing	126
6.3	Preliminary	127
6.3.1	Probabilistic Approach to Mesh Connectivity	127
6.3.2	Basic Principles	128
6.4	Formulation	132
6.4.1	Practical approach to compute Λ_{wdt}	132
6.4.2	Definition of Λ_{real}	133
6.4.3	Loss Functions	134
6.5	Experiments and Applications	136
6.5.1	Mesh to DMesh	136
6.5.2	Point Cloud & Multi-View Reconstruction	137
6.6	Conclusion	140
6.6.1	Limitations	140
6.6.2	Future Work	142
Chapter 7:	DMesh++: An Efficient Differentiable Mesh for Complex Shapes	143
7.1	Introduction	144
7.2	Related Work	146
7.3	Formulation	148

7.3.1	Preliminary	148
7.3.2	Minimum-Ball Algorithm	150
7.4	Reconstruction Process	152
7.5	Experiments	153
7.5.1	Tessellation Cost	154
7.5.2	Reconstruction Tasks	155
7.6	Conclusion	163
Chapter 8:	Differentiable Delaunay Tetrahedral Mesh Generation	165
8.1	Introduction	166
8.2	Related Work	168
8.2.1	Tetrahedral Meshing	168
8.2.2	Differentiable Mesh	169
8.3	Approach	170
8.3.1	Definition	170
8.3.2	Overview	171
8.3.3	Volumetric Marching Tetrahedra (MT)	172
8.3.4	Loss Formulation	175
8.4	Experimental Results	179
8.4.1	Metrics	179
8.4.2	SDF Conditioned Mesh Optimization	180
8.4.3	Mesh Reconstruction from Multi-View Images	182
8.4.4	Validation with Physics Simulation	184
8.5	Conclusion	184
V	Conclusion	187
Chapter 9:	Conclusion	188
9.1	Summary of Results	188
9.2	Future Work	190
VI	Appendix	193
Appendix A:	Gradient Informed Proximal Policy Optimization	194
A.1	Proofs	194
A.1.1	Proof of Lemma 2.3.2	194
A.1.2	Proof of Lemma 2.4.2	195
A.1.3	Proof of Proposition 2.4.3	196
A.1.4	Proof of Lemma 2.4.4	197
A.1.5	Proof of Lemma 2.4.6	197
A.1.6	Proof of Proposition 2.4.5	198
A.2	Formulations	199
A.2.1	Analytical Gradient of Generalized Advantage Estimator (GAE)	199

A.2.2	RP Gradient Formulation	200
A.2.3	Estimator for Equation 2.4	201
A.3	Algorithm	202
A.3.1	Empirical evidence	202
A.3.2	Loss function for PPO Update	203
A.3.3	Example: Function Optimization Problems (Section 2.5.1)	204
A.3.4	Example: Physics Simulation Problems (Section 2.5.2)	206
A.4	Experimental Details	206
A.4.1	Baseline Methods	206
A.4.2	Network architecture and Hyperparameters	209
A.5	Problem Definitions	216
A.5.1	Function Optimization Problems (Section 2.5.1)	216
A.5.2	Traffic Problems (Section 2.5.3)	218
A.6	Renderings	221
A.6.1	Traffic Simulation	221
A.7	Computational Costs	223
Appendix B: Time-Aware World Model for Adaptive Prediction and Control		225
B.1	Descriptions of 4th-order Runge-Kutta integration.	225
B.2	Meta-World Task Dynamics Visualizations.	226
B.3	Additional results for ablation study of Δt sampling strategies.	226
B.4	Comparisons with Multi Time Scale World Model (MTS3).	229
B.5	Additional performance comparisons between TAWM and baseline models	232
B.6	Additional learning curve comparisons between TAWM and baseline models	233
B.7	Additional experiments on PDE control problems.	236
B.7.1	PDE Control Problems.	236
B.7.2	Burgers' Equation.	237
B.7.3	Allen-Cahn Equation.	238
B.7.4	Wave Equation.	238
B.7.5	Additional learning curve comparisons on PDE control tasks.	239
B.7.6	Proofs	241
Appendix C: Differentiable Hybrid Traffic Simulation		243
C.1	Solution to the Riemann problem in ARZ model	243
C.1.1	Formulation	243
C.1.2	Differentiation	245
C.2	Differentiating IDM	248
C.2.1	Hyperparameters	248
C.2.2	Differentiation	249
Appendix D: Gradient-based Trajectory Optimization with Parallelized Differentiable Traffic Simulation		251
D.1	Feature Comparison with Existing Vehicle Simulators	251
Appendix E: DMesh: A Differentiable Mesh Representation		252

E.1	Comparison to Other Shape Reconstruction Methods	252
E.2	Details about Section 6.3.2	254
E.2.1	Mathematical Definitions	254
E.2.2	Analysis of previous approach	256
E.3	Loss Functions	260
E.3.1	Mesh to DMesh	260
E.3.2	Point Cloud Reconstruction	261
E.3.3	Multi-View Reconstruction	266
E.3.4	Weight Regularization	271
E.3.5	Real Regularization	272
E.3.6	Quality Regularization	273
E.4	Optimization Process	273
E.4.1	Mesh to DMesh	274
E.4.2	Point cloud & Multi-view Reconstruction	276
E.4.3	Post-Processing	281
E.5	Experimental Details	282
E.5.1	Mesh to DMesh	282
E.5.2	Point Cloud & Multi-view Reconstruction	284
E.5.3	Ablation studies	287
Appendix F:	DMesh++: An Efficient Differentiable Mesh for Complex Shapes	294
F.1	Details about <i>Minimum-Ball</i> algorithm	294
F.1.1	Algorithm	294
F.1.2	Minimum-Ball computation	295
F.1.3	Sigmoid coefficient α_{min}	296
F.1.4	Nearest neighbor caching	298
F.2	Details about Reconstruction Process	298
F.2.1	Loss Formulation	299
F.2.2	Reconstruction Steps	302
F.3	Experimental Details and Additional Results	308
F.3.1	Dataset	308
F.3.2	2D Point Cloud Reconstruction	309
F.3.3	3D Point Cloud Reconstruction	310
F.3.4	3D Multi-View Reconstruction	311
F.4	<i>Reinforce-Ball</i> algorithm	316
F.4.1	Local Minima of Weight Regularization	316
F.4.2	Algorithm Overview	319
F.4.3	Formal Definition	320
F.4.4	Experimental Results	323
Appendix G:	Differentiable Delaunay Tetrahedral Mesh Generation	325
G.1	Volumetric Marching Tetrahedra Algorithm	325
Bibliography		326

List of Tables

2.1	Average maximum reward ($\uparrow$) for function optimization problems.	42
4.1	Comparison of our technique against automatic differentiation for various types of simulations. Scale denotes number of time steps taken and the number of cells (for macroscopic, and hybrid)/ number of vehicles (microscopic). FW: forward simulation; BP: backpropagation. Ours outperforms autodifferentiation in both forward passes and backpropagation, by up to a factor of 18.77 , 7.84 , and 15.27 in <i>macroscopic</i> , <i>microscopic</i> , and <i>hybrid</i> traffic simulation, respectively.	90
4.2	Comparison of maximum rewards from each control problem. Our framework is able to find better solutions (indicated by higher reward values) than the other baseline, model-free RL algorithms (DDPG, PPO, or SAC) for every experiment setting (Macro, Micro, or Hybrid).	97
5.1	Quantitative comparison for trajectory filtering & reconstruction task. For each task, we compare the average positional error rate (Pos.), magnitude of estimated acceleration (Acc.), physically implausible trajectory rate (Imp.), and average computation time per trajectory (Time.) between compared methods. For estimated acceleration magnitude (m/s^2), we provide mean and standard variation (μ/σ).	112
5.2	Waymo Open Motion Dataset validation set metric comparisons between SOTA models and training-free rollouts generated by our differentiable simulator. We show metric performance results generated from our simulator versus current SOTA models for trajectory prediction. Without any neural networks or training, future predictions using our simulator can attain 0.3541 mAP on validation—while this result does not outperform SOTA models, this performance is attained with <i>no training</i> , where IDM parameters are only fitted on 1-second of data . Missing values are not reported by the respective original work.	119
6.1	Mesh reconstruction results.	137
6.2	Quantitative comparison for point cloud and multi-view reconstruction results. Best results are written in bold.	139
7.1	Quantitative comparison of the 2D point cloud reconstruction task for the font dataset. DMesh++ reconstructs 2D meshes with greater accuracy and efficiency than DMesh.	156
7.2	Quantitative comparison of 3D point cloud reconstruction results over 50 manually chosen models from Thingi10K [1] and Objaverse [2]. We highlight the best results for each metric. The upper panel compares methods that use <i>unoriented</i> point clouds (VoroMesh, DMesh++), while the lower panel displays methods that use <i>oriented</i> point clouds (PSR, PoNQ, DMesh, DMesh++).	158
7.3	Quantitative comparison of 3D point cloud reconstruction results over 500 randomly chosen models from Thingi10K [1]. The results over closed surfaces and open surfaces are shown together: (closed / open). We highlight the best results for two different categories with red and blue , respectively.	158

7.4	Quantitative comparison of 3D multi-view reconstruction results over 50 manually chosen models from Thingi10K [1] and Objaverse [2]. We highlight the best results and the second best results for each evaluation metric.	160
7.5	Quantitative comparison of 3D multi-view reconstruction results over 500 randomly chosen models from Thingi10K [1]. The results over closed surfaces and open surfaces are shown together: (closed / open). We highlight the best results for two different categories with red and blue, respectively.	161
8.1	Quantitative Comparison of Extracted Mesh	180
8.2	Quantitative comparison of the multi-view reconstruction results. For each metric, the better result is highlighted with color.	182
A.1	Average (wall clock) training time (sec) for traffic problems.	224
D.1	Feature comparisons between existing popular traffic simulators and ours. Note that this comparison is for vehicle simulation only.	251
E.1	Traits of different optimization-based shape reconstruction methods. We compare methods based on template mesh [3, 4], neural SDF [5, 6], neural UDF [7, 8], differentiable isosurface extraction techniques [9–12] with ours.	253
E.2	Comparison of computational speed and accuracy between the previous method [13] and ours. For each number of points in 2D, we report the triplet of (computational speed (ms) / false positive ratio (%) / false negative ratio (%)), along with the number of query simplexes that we give to the algorithm.	259
E.3	Ablation study for weight regularization, quantitative results.	288
E.4	Ablation study for quality regularization, quantitative results.	290
F.1	Quantitative ablation studies on Reinforce-Ball algorithm. As we increase ϵ_{card} (in parenthesis) for DMesh++, we can significantly reduce the mesh complexity without losing geometric details, while DMesh cannot do the same with λ_{weight}	322

List of Figures

1.1	The overview of my research. The upper box shows the keyword, and the lower block shows specific work.	4
2.1	α -policies for 1-dimensional (a) De Jong’s function and (b) Ackley’s function. The original policy is rendered in blue, and alpha policies for different α values are rendered in orange and red. Those α -policies are obtained from analytical gradients of the given function $f(x)$ rendered in black. Note that Ackley’s α -policy becomes invalid when $\alpha = 2 \cdot 10^{-3}$, because of singularities near 0.	35
2.2	Optimization curves for Dejong’s and Ackley’s function of dimension 1 and 64.	42
2.3	Learning graphs for 3 problems in differentiable physics simulation: Cartpole, Ant, and Hopper [14]. In these environments, RP gradients already exhibit lower variance without much bias than LR gradients, because of variance reduction scheme of [14].	44
2.4	Learning graphs for traffic pace car problem. Each problem has different number of lanes and following human vehicles, which represent <i>discontinuities</i> and <i>dimension of the problem</i> , respectively.	45
3.1	Overall framework of our time-aware world model. An encoder h encodes the given observation o_t into a latent vector z_t , which is then fed into various models with action a_t and <i>time step size</i> Δt to estimate values for action planning.	50
3.2	Visualizations of the Meta-World control tasks. From left to right: <i>Assembly</i> , <i>Basketball</i> , <i>Box Close</i> , <i>Faucet Open</i> . The default time step size of all environments is $\Delta t = 2.5ms$	65
3.3	Performance comparisons for 3 tasks from Meta-World (Up), and 3 tasks from PDE-Control (Down). The x-axis corresponds to the evaluation Δt (in <i>ms</i>), and the y-axis corresponds to success rate (Meta-world) and episode reward (PDE-Control). We trained our TAWM model using either RK4 or Euler integration method with log-uniform sampling strategy. The baseline method was trained using only default time step ($\Delta t = 2.5ms$ for Meta-World tasks; $\Delta t = 50ms/10ms/100ms$ for PDE-Burgers, PDE-Allen-Cahn, PDE-Wave, respectively), which is signified with the black dashed line in each environment. For fair comparisons, we extended the baseline method by repeating the same actions for the larger evaluation time steps than the default one. For instance, when the evaluation $\Delta t = 5ms$, we repeat the same action for $5/2.5 = 2$ times. Plots show mean and 95% confidence intervals over 3 seeds, with 10 evaluation episodes per seed.	66

3.4	Performance comparisons to baseline methods trained on different Δt values for Meta-World tasks. The x-axis corresponds to the evaluation Δt , and the y-axis corresponds to the success rate. The non-time-aware baseline models were trained with different fixed Δt values (shown in the legend). Our TAWM model employs RK4 integration and log-uniform Δt sampling. TAWM outperforms all baseline models trained with fixed time-step sizes. When the baselines are trained only at low observation rates ($\Delta t \geq 10\text{ms}$), they fail on <i>all</i> tasks. The dashed black line indicates the default time-step size ($\Delta t = 2.5\text{ms}$). Means and 95% confidence intervals are computed over 3 seeds, each evaluated for 10 episodes.	67
3.5	Learning curve comparison on Meta-World Basketball (Up) and Meta-World Lever-Pull (Down). The x-axis corresponds to the evaluation Δt (in <i>ms</i>), and the y-axis corresponds to the success rate. We trained our TAWM using either RK4 or Euler integration method with log-uniform sampling strategy. The baseline method was trained using only the default time step ($\Delta t = 2.5\text{ms}$). Plots show mean and 95% confidence intervals over 3 seeds, with 10 evaluation episodes per seed.	68
3.6	Performance comparisons to MTS3 across different evaluation Δt's on Meta-World tasks. The x-axis corresponds to the evaluation Δt , and the y-axis corresponds to the success rate. The MTS3 models were trained under different slow dynamics settings, indicated by the H values shown in the legend, using offline data with 4M transitions. Our TAWM employs RK4 integration method with log-uniform sampling strategy and was trained for 1.5M steps. The dashed black line indicates the default time-step size ($\Delta t = 2.5\text{ms}$). Means and 95% confidence intervals are computed over 3 seeds for TAWM, each evaluated for 10 episodes. TAWM outperforms MTS3 across different H settings, especially at large Δt values.	69
3.7	Ablation Study on Δt Sampling Strategy. We trained our TAWM models with log-uniform and uniform sampling strategies, both of which use the RK4 integration method. The baseline method was trained using only default time step ($\Delta t = 2.5\text{ms}$), which is signified with the black dashed lines. The x-axis corresponds to the evaluation Δt , and the y-axis corresponds to the success rate. Means and 95% confidence intervals are computed over 3 seeds, each evaluated for 10 episodes.	70
4.1	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. (<i>This scene is rendered in CARLA [15].</i>)	75
4.2	Comparison on the average frame rate (FPS) under the hybrid model, with different ϵ . The total number of vehicles is set to be 10K, and the ϵ value denotes the ratio of microscopic vehicles among them.	89
4.3	Parameter estimation problem. For given final states of the system, one has to estimate the initial states that lead to the final states. (a)–(d) Our results rendered in blue exhibit much faster convergence to the correct solutions than the other state-of-the-art gradient-free optimization methods by up to <i>one order of magnitude</i>	91

4.4	4x4 Grid of Intersections. Hybrid traffic simulation at a 4x4 grid of intersections: the density of each cell is rendered in red, which shows the volume of traffic agents to cross the intersection. One has to optimize time allocations for traffic lights at each intersection to maximize overall traffic flow.	93
4.5	ITSCP. (a), (b) Learning graphs for macroscopic and hybrid simulations. Our framework shows better convergence rates and results than other RL algorithms.	94
4.6	Pace Car Problem. Sequence of simulation images in time order, from top to bottom. (a) A pace car rendered in green guides following vehicles to maintain a target speed limit, which varies over time. It fails when it runs too fast, too slow, or goes out of the lane. (b) When a high target speed limit (30 m/sec) is given, the pace car accelerates fast, and keeps a wide distance from its following vehicles to let them run fast. (c) However, when the target speed limit is reduced (10 m/sec), it swiftly decelerates and keeps short distance from the following vehicles, to prevent them from running over the target speed limit.	94
4.7	Pace Car Learning Graph. Our result rendered in blue shows far better performance (as indicated by higher reward values) than other algorithms, which make no significant gains in learning even after long periods.	96
4.8	Large-scale Hybrid Traffic simulation. These traffic scenes consist of many hundreds of vehicles across an urban scene simulated and controlled in real time using a differentiable hybrid traffic model.	99
5.1	Computational cost of our traffic simulator. Using either (multi-threaded) CPU or GPU, our simulation takes less than 30 milliseconds <i>per timestep</i> to process 2 million vehicles in both (a) forward and (b) backward pass.	102
5.2	Trajectory Optimization Problems. For the given data points (red flags), we can fit a simulated trajectory to them by optimizing IDM variables. (a) When data points are dense, we can filter physically inaccurate noises in the original trajectory. (b) When they are sparse, we can reconstruct dense trajectories. (c) We can even predict future trajectories based on the IDM variables.	105
5.3	Overall Framework. In our traffic simulator, for each frame, (a) we first collect variables for each vehicle in the scene to use in IDM. These variables include the position (p_i) and velocity (v_i) of each vehicle, and its relationship to its leading vehicle ($p_{h(i)}$, $v_{h(i)}$). Since we can apply IDM to each vehicle's variables independently, we can process them in parallel, rather than sequentially. In (b) and (c), the process ordering is depicted with colors and numbers. In (c), we assume there are two computational units that can run in parallel.	106
5.4	Qualitative comparison for trajectory filtering task. For comparison, we use the speed (up) and acceleration (down) profiles for a single trajectory in the NGSIM dataset. Estimated acceleration from our method exhibits a more stable pattern than the baseline methods.	114
5.5	Distribution of optimized IDM parameters in trajectory reconstruction task. Note that the distribution of each parameter adheres well to the general, real-world scenarios.	116

5.6	Training-free future rollouts on the Waymo Open Motion Dataset (WOMD). We show qualitative results in trajectory forecasting with our differentiable traffic simulator. In the trajectory forecasting task, we infer the traffic state based on road graph and agent observations, including agent membership and order within lanes. Then, we fit IDM parameters for each vehicle to its 1-second historical trajectory and roll out trajectories via IDM for all agents according to their current lane. As shown above, agent vehicles maintain realistic trajectories with no deep learning or ground truth. . . .	117
6.1	($\rightarrow$) Optimization process. We can optimize our mesh starting from either (a) random state or (b) initialization based on sample points for faster convergence. Mesh connectivity changes dynamically during the optimization. To make this topology change possible, we compute existence probability for an arbitrary set of faces in a differentiable manner.	122
6.2	Versatility of DMesh. DMesh can represent diverse geometry in differentiable manner, including (a) non-convex polyhedra of different Euler characteristics, (b) non-orientable geometries (Möbius strip, Klein bottle), and (c) complex protein structure (colored for aesthetic purpose).	122
6.3	Our overall framework to optimize mesh according to the given observations. (a): Each point is defined by a 5-dimensional feature vector: position, weight, and real value. Points with larger real values are rendered in red. (b): Given a set of points, we gather possibly existing faces in the mesh and evaluate their probability in differentiable manner. (c): We can compute reconstruction loss based on given observations, such as mesh, point cloud, or multi-view images. (d): To facilitate the optimization process and enhance the mesh quality, we can use additional regularizations.	125
6.4	Illustration of our mesh representation for 2D and 3D cases. (a): Our representation in 2D for a letter “A”. (b): Our representation in 3D for a dragon model. Blue faces are “ <i>real part</i> ” and yellow ones are “ <i>imaginary part</i> ”.	127
6.5	Renderings of Δ^k s for different pairs of (d, k) . Different Δ^k s are rendered in different colors.	129
6.6	Basic concepts to compute existence probability of given 1-simplex (Δ^1) when $d = 2$. (a): WDT and PD of given set of weighted vertices are rendered in solid and dotted lines. The size of a vertex represents its weight. (b): Power cell of p_1 (C_{p_1}) is rendered in grey. Also, Δ^1 is rendered in black line, of which dual line (D_{Δ^1}) is rendered in red. (c), (d): For given Δ^1 , reduced power cell of p_1 for the Δ^1 ($R_{p_1 \Delta^1}$) is rendered in blue, with the original power cell (grey). We can evaluate the existence of Δ^1 in WDT by computing the signed distance from D_{Δ^1} to $R_{p_1 \Delta^1}$	130
6.7	Results with different λ_{weight}.	135
6.8	Point cloud and multi-view reconstruction results. (a): Ground truth mesh. (b), (f): Our method restores the original shape without losing much detail. (c), (d), (g), (h): PSR [16], VoroMesh [17], FlexiCube [11], and NIE [18] fail for open and mixed surfaces. (e): NDC [19] exhibits artifacts from grids.	138
6.9	($\rightarrow$) Shape interpolation using DMesh exhibiting topology change. After fitting DMesh to a torus (upper left), we optimize it again to reconstruct a double torus (lower right), which has a different genus. We use multi-view images for the optimization. . . .	140

6.10	Analysis of computational cost for computing face existence probabilities ($\Lambda(F)$). The computational cost rises sharply beyond 20K points, with most of the time spent on WDT construction (“WDT”), while the probability computation (“Prob”) requires significantly less time.	141
7.1	DMesh++ for complex 2D and 3D shapes. DMesh++ encodes all geometric and topological information into continuous point features. (a) By optimizing these point features, DMesh++ is able to reconstruct complex 2D drawings from sample points. (b) This approach is also applicable to 3D, where it reconstructs the complex geometric structure of DNA from a point cloud. (c) By incorporating additional color features, DMesh++ can reconstruct complex, colored 3D shapes from multi-view images. For each result, the “ <i>imaginary</i> ” part is rendered in gray, while the “ <i>real</i> ” part—which defines the final mesh—is rendered in other colors. (section 7.3.1).	143
7.2	Conceptual comparison of traditional mesh and variants of DMesh [20]. Traditional meshes employ a non-differentiable, discrete data structure, $\mathbb{F}$, to store vertex indices that define connectivity, whereas DMesh++ encodes connectivity and additional information into continuous point-wise features, $\mathbb{P}$. Mesh generated from DMesh++ avoids several degeneracies—such as self-intersections and thin triangles—that can compromise its suitability for downstream applications.	145
7.3	Mesh topology change during 3D point cloud reconstruction of a vase. We visualize the evolution of point-wise features by projecting them onto a 2-dimensional space using T-SNE [21] at each optimization step (ranging from 0 to 2K). As these features continuously evolve, the mesh undergoes discrete topological changes to progressively recover the target shape.	146
7.4	Bounding balls of a face F (red) in 2D (left) and 3D (right). The minimum bounding ball (B_F) is rendered in blue, while the others are rendered in gray.	148
7.5	Minimum-Ball condition in 2D. In the left, 2D Delaunay Triangulation (DT) of 6 points is given. In middle and right figure, we render B_F for two faces ($\overline{AB}$, $\overline{DF}$) in blue.	150
7.6	Reconstruction process for 3D multi-view colored images of a sculpture. In each stage, we optimize different per-point features: the position and the real (ψ), while the per-point color is refined at every stage. (Left) We display the meshes at each stage during the first epoch. (Right) We provide a rendering of the final mesh along with its view-point normal, which reveals the fine mesh details.	153
7.7	Comparison of tessellation cost. Our method computes face probabilities up to 16 times faster in 2D and 32 times faster in 3D than DMesh [20], while using up to 96% less GPU memory in 2D and 75% less in 3D.	154
7.8	Qualitative comparison of 2D point cloud reconstruction results. The outputs of DMesh [20] and DMesh++ are rendered in red and blue, respectively.	156
7.9	Qualitative comparison of 3D point cloud reconstruction results for a toad sitting on a leaf. For each result, we render its diffuse image on the left, and view-point normals on the right.	159
7.10	Qualitative comparison of 3D multi-view reconstruction results for open surface. Here we illustrate from back of an open surface model (the front view is rendered at the left top of (a)). Colors represent inside and outside facing surfaces	160

7.11	Self-intersection of the reconstructed mesh. The self-intersected faces of the mesh are rendered in red.	161
7.12	Physics simulation on a staircase reconstructed from multi-view images. We simulate the motion of bouncing balls directly on the mesh generated by DMesh++. . . .	163
8.1	Tetrahedral mesh reconstruction from multi-view images. Using our differentiable tetrahedral mesh formulation, we reconstruct a high-quality volumetric mesh from multi-view images. Over the course of optimization, geometric fidelity and mesh quality improve concurrently: the recovered surface resolves fine-scale image details, and the tetrahedral elements and surface triangles become more regular. Notably, the fraction of elements whose minimum dihedral angle is below 10° (shown in red) decreases markedly. The final mesh further exhibits internal grading: fine near the boundary and coarse in the interior; thereby concentrating degrees of freedom where they matter most and yielding a more efficient representation without losing precision.	165
8.2	Comparison of reconstructed tetrahedral mesh to FlexiCubes [11]. Compared with FlexiCubes, our method yields more element-efficient, internally graded tetrahedral mesh as it allows topological tessellation changes.	167
8.3	Overview of our tetrahedral mesh optimization pipeline (illustrated in 2D). (a) Initialize the vertex set: retain samples near the zero level set (red) and discard far-away samples (black). (b) Obtain the tetrahedral grid by computing a Delaunay triangulation (DT) of the retained vertices. (c) Using this grid and per-vertex SDF values, extract a volumetric tetrahedral mesh via a volumetric Marching Tetrahedra (MT) algorithm. Colors encode mesh quality; red triangles indicate poor aspect ratios. (d) Improve quality by minimizing losses defined on the extracted mesh and backpropagating to per-vertex attributes (positions and SDF values). After several iterations, the vertex features are optimized to produce an element-efficient, high-quality tetrahedral mesh. .	171
8.4	Tetrahedron classification for volumetric mesh extraction. Based on vertex signs, we can broadly classify a tetrahedron T into 3 cases. The sign of each vertex is labeled as + (outside) or - (inside), and a boundary surface patch of the target shape and its outward normal is rendered together. (a) When every vertex is outside, we can exclude T from our mesh. (b) When the boundary of the target shape intersects the tetrahedron, some vertices are outside while the others are inside. Then, we need to extract subset of T that is inside the target shape (see fig. 8.5b). (c) When every vertex is inside, we can include T in our mesh.	172
8.5	Inner part approximation. For each edge that connects vertices that have different signs, we can approximate the point on the iso-surface using linear interpolation, which are rendered in green. By connecting red inner vertices and these approximation points, we can get the orange convex polyhedron that approximates the inner part of the tetrahedron.	174
8.6	Tetrahedral decomposition of a polyhedron. (a) We decompose a polyhedron using a centroid C . (b) For each face T on the polyhedron, if it is not a triangle (e.g., T_b , T_c), we subdivide the face into a set of triangles (e.g., $T_b \rightarrow T_{ba}$, T_{bc}). (c) We can define a set of tetrahedra that compactly decomposes the polyhedron by connecting each triangle with C . The tetrahedra not visible from the front are not rendered. . . .	175

8.7	Effect of quality loss in differentiability. (a) We can attain better reconstruction loss when we use quality loss. (b) This is because it contributes to suppressing discontinuous topological changes, which could harm the optimization process.	177
8.8	Qualitative comparison of the volumetric mesh representing mechanical structures. Using our method, we can attain geometrically accurate mesh comparable to MC [22] that used $2\times$ higher grid resolution, while the baseline MT [23] is noticeably less accurate. Our mesh is comparably more element-efficient than MT, as ours prunes away unnecessary vertices.	181
8.9	Qualitative comparison of the reconstruction results from multi-view images (From top: Knot, Hand, and Bunny) Compared to FlexiCubes [11], our method reconstructs much more element-efficient mesh, without losing visual details and mesh quality. . . .	183
8.10	Physics simulation using reconstructed Armadillo. We pass the Delaunay-triangulated output of our method directly to an elastic FEM simulator with contact. Rigid balls are dropped and collide with the deformable body, producing contact-rich behavior. The full animation is provided in the supplementary video.	184
A.1	The first row shows the sample variance of analytical gradients ($\frac{\partial A}{\partial a}$), and the second row shows the statistics of estimates of $\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))$ for every state-action pair in the buffer after each epoch of training in 3 different differentiable physics environments (Cartpole, Ant, and Hopper). We used a fixed $\alpha = 10^{-2}$. Note that the minimum and maximum estimates in the second row exhibit a clear tendency to deviate from 1 more as the sample variance increases.	203
A.2	Change of α during solving function optimization problems in Section 2.5.1. Since these problems are one-step problems, and observations are all the same, there is no difference between mean, min, and max of estimated determinant.	204
A.3	Change of α during training in Ant environment in Section 2.5.2. In this environment, maximum out-of-range-ratio (δ_{oorr}) plays a major role in bounding α from growing. Observe that GI-PPO achieves better results than RP when we set $\delta_{oorr} = 1.0$	205
A.4	Landscape of target functions in 2 dimensions.	217
A.5	Traffic Environment. In traffic flows, only partial gradients of forward simulation are known; acceleration along a (longitudinal) traffic lane is continuous and admits gradient flow (green), while (lateral) lane-changing is a discrete event and thus prohibits gradient flow (red). Analytical gradients convey only partial information about traffic dynamics.	222
A.6	10-Lane pace car environment. The autonomous vehicle (Blue) has to interfere paths of the other human driven vehicles (Yellow) to limit their speeds to $10m/s$. Even though it is a hard problem to achieve high score, our method achieved far better score than the baseline methods.	223
B.1	Ablation Study on Δt Sampling Strategy. We trained our TAWM models with log-uniform and uniform sampling strategies, both of which use the RK4 integration method. The baseline method was trained using only default time step ($\Delta t = 2.5ms$), which is signified with the black dashed lines. The x-axis corresponds to the evaluation Δt , and the y-axis corresponds to the success rate. Means and 95% confidence intervals are computed over 3 seeds, each evaluated for 10 episodes.	226

B.2	Visualizations of the Meta-World control tasks. From top to bottom: (1) <i>Assembly</i> , (2) <i>Basketball</i> , (3) <i>Box Close</i> , (4) <i>Faucet Open</i> , (5) <i>Hammer</i> . From left to right: sequences of renderings from task initialization to completion.	227
B.3	Visualizations of the Meta-World control tasks. From top to bottom: (6) <i>Handle Pull</i> , (7) <i>Lever Pull</i> , (8) <i>Pick Out Of Hole</i> , (9) <i>Sweep Into</i> . From left to right: sequences of renderings from task initialization to completion.	228
B.4	Performance comparisons for 6 additional tasks from Meta-World tasks. The x-axis corresponds to the evaluation Δt (in <i>ms</i>), and the y-axis corresponds to success rate. We trained our TAWM model using either RK4 or Euler integration method with log-uniform sampling strategy. The baseline method was trained using only the default time step ($\Delta t = 2.5ms$), which is signified with the black dashed lines. For fair comparisons, we extended the baseline method by repeating the same actions for the larger evaluation time steps than the default one. For instance, when the evaluation $\Delta t = 5ms$, we repeat the same action for $5/2.5 = 2$ times. Plots show mean and 95% confidence intervals over 3 seeds, with 10 evaluation episodes per seed.	232
B.5	Additional Meta-World tasks: Success Rate Curve under different evaluation time step sizes.	234
B.6	Additional Meta-World tasks: Success Rate Curve under different evaluation time step sizes.	235
B.7	Visualizations of systems dynamics of uncontrolled Burgers PDE. The spatial domain has length $L = 1$ with the diffusivity (viscosity) parameter $\nu = 10^{-3}$. The initial state is $u(x, t = 0) = sech(10x - 5)$	237
B.8	Visualizations of systems dynamics of uncontrolled Allen-Cahn PDE. The spatial domain has length $L = 2$ with the diffusivity (viscosity) parameter $\nu = 10^{-4}$ and potential constant $V = 5.0$. The initial state is $u(x, t = 0) = (x - 1)^2 \cdot cos(\pi(x - 1))$	238
B.9	Visualizations of systems dynamics of uncontrolled Wave PDE. The spatial domain has length $L = 1$ with $c = 0.1$. The initial state is $u(x, t = 0) = sech(10x - 5)$ and $\psi(x, t = 0) = 0$	239
B.10	PDE control tasks: Average Negative LQ Error learning curves under different evaluation time step sizes. The default time step size are $\Delta t = 50ms/10ms/100ms$ for PDE-Burgers, PDE-Allen-Cahn, PDE-Wave, respectively.	240
E.1	$\mathcal{F}_B$ uses tessellation structure to efficiently render overlapped faces in the correct order.	268
E.2	Rendered images from two differentiable renderers, $\mathcal{F}_A$ and $\mathcal{F}_{A'}$. Left and right image corresponds to diffuse and depth rendering, respectively. (a) $\mathcal{F}_A$ is our (partially) differentiable renderer based on tile-based approach. (b) Since $\mathcal{F}_A$ does not produce visibility-related gradients, we additionally use $\mathcal{F}_{A'}$ [24] to render images and integrate with ours.	269
E.3	Reconstructed mesh from multi-view images, rendered in MeshLab’s [25] x-ray mode to see inner structure. In multi-view reconstruction, we divide each epoch in two phases. (a) After the first phase ends, where we do inaccurate depth testing, lots of false inner faces are created. (b) To remove these inner faces, we require a renderer that does the exact depth testing, which we use in the second phase. Also see Appendix E.4.2 for details about post-processing step to remove the inner structure.	270

E.4	Intermediate results in converting bunny model to DMesh. For given ground truth mesh in (a), we initialize our point attributes using the mesh vertices. (b) Then, the initial mesh becomes convex hull of the original mesh. (c) To remove undesirable faces that were not in the original mesh, we insert additional points on the undesirable faces. Then, some of them disappear because of the inserted points. (d) After optimizing 5000 steps, just before another point insertion, DMesh recovers most of the ground truth connectivity.	275
E.5	Initialized DMesh using sample points from ground truth mesh. (a) From ground truth mesh, we uniformly sample $10K$ points to initialize DMesh. (b) In the left figure, sample points from the ground truth mesh ($\mathbb{P}_{sample}$) are rendered in red. The points that correspond to $\mathbb{P}_{voronoi}$ are rendered in blue. In the right figure, we render the initial mesh we can get from the points, which has a lot of holes.	279
E.6	Optimization process for multi-view reconstruction for Plant model. At each row, we present the initial state (left) and the last state (right) of each epoch. For each figure, the left rendering shows the point attributes color coded based on real values, while the right one shows the extracted mesh. (a), (b) In the first epoch, we initialize DMesh without sample points. At the end of each epoch, we sample points from the current mesh, and use them for initialization in the next epoch.	280
E.7	Reconstruction results for mesh to DMesh experiment. From Left: Bunny, Dragon, and Buddha. We can observe that most of the edge connectivity is perserved in the reconstruction, even though the appearance is slightly different from the ground truth mesh because of small perturbations of vertex positions.	283
E.8	Reconstruction results for a closed surface model in Thingi32 dataset. Flexicube [11] can generate internal structures, while our approach removes them through post-processing.	285
E.9	Reconstruction results for the Plant model. Flexicube [11] can generate redundant, self-intersecting faces for open surfaces, in this case, leaves. To better capture the redundant faces, we rendered the models from upper side, which is shown in the bottom right figures.	286
E.10	Point cloud reconstruction results with different λ_{weight}. From Left: $\lambda_{weight} = 10^{-6}, 10^{-5}$, and 10^{-4}	288
E.11	Point cloud reconstruction results with different $\lambda_{quality}$. From Left: $\lambda_{real} = 10^{-4}, 10^{-3}$, and 10^{-2}	289
E.12	Point cloud and Multi-view Reconstruction results for open surface models. From Left: Ground truth mesh, sample point cloud, point cloud reconstruction results, diffuse rendering, multi-view reconstruction results.	291
E.13	Point cloud and Multi-view Reconstruction results for closed surface models. From Left: Ground truth mesh, sample point cloud, point cloud reconstruction results, diffuse rendering, multi-view reconstruction results.	292
E.14	Point cloud and Multi-view Reconstruction results for mixed surface models. From Left: Ground truth mesh, sample point cloud, point cloud reconstruction results, diffuse rendering, multi-view reconstruction results.	293

F.1	Common signed distance for a 2D (left) and 3D (right) face in (initial) regular grid. We compute the signed distance by subtracting the radius of the minimum bounding ball from the length of the red line. The red dot represents the center of the minimum bounding ball.	296
F.2	Role of visibility gradient in geometric optimization. In this experiment, we optimize the translation vector of the object by comparing its rendered image and the ground truth image on the left. Since the differentiable renderer of DMesh [20] does not implement visibility gradient, while ours does, DMesh fails to find the correct translation vector.	299
F.3	Implementation of anti-aliasing in our differentiable renderer. On the left, we show the process of anti-aliasing: 1) Find pixels that overlap with the given triangle, 2) Find the area that each pixel overlaps with the given triangle, and 3) Determine the color of each pixel based on the overlap area. On the right, we show details of the step 2.	301
F.4	Grid structure to initialize real values in 2D (left) and 3D (right). Every face in the grid structure satisfies <i>Minimum-Ball</i> condition (Definition 7.3.1).	304
F.5	Point insertion for removing undesirable face. (Left) To reconstruct the ground truth shape, we need to set the real value (ψ) of points A-E to 1. The point rendered with dotted line has real value of 0. Then, we observe unnecessary face $\overline{BD}$ exists. (Right) To remove this face, we insert additional point that carries 0 real value near the unnecessary face.	307
F.6	2D point cloud reconstruction result for complex drawings. For each drawing, we report both the number of edges and the reconstruction time. For the Chinese drawing, we additionally render the “imaginary” part on the right to clearly illustrate its complexity.	310
F.7	Qualitative comparison of 3D point cloud reconstruction results for a closed surface (vase). For each image, we render the view-point normal on the right, and the diffuse image on the left. Among the baseline methods that reconstruct watertight mesh from point clouds, PoNQ [26] performs the best in reconstructing fine geometric details. While DMesh [20] fails at reconstructing such details due to the lack of mesh complexity, DMesh++ successfully recovers them and produce comparable result to PoNQ.	311
F.8	Qualitative comparison of 3D multi-view reconstruction results for a closed surface (sculpture). We render the input diffuse and depth images alongside the ground truth image. For each image, we render the view-point normal on the left, and the diffuse image on the right. We can observe that the reconstruction result of DMesh++ is as good as the other baseline methods that are optimized for closed surfaces.	312
F.9	3D reconstruction from real-world images in DTU dataset [27]. The input images are shown on left, and the reconstructed mesh is shown on right.	315

F.10	Local minima of weight regularization in a rendering setting. (1) The ground truth geometry is rendered in gray dotted line. There are 3 points (A, B, C), where only the end points (A, B) are necessary for fully representing the underlying shape. Every point has weight 1.0, which is written to the next of each point. In this case, faces $\overline{AC}$ and $\overline{BC}$ exist with probability 1.0, which corresponds to their opacity. In this case, for a ray that goes through this mesh, the accumulated opacity (A.O.) becomes 1.0, and the reconstruction loss is 0. (2) When weight regularization reduces the weight of (redundant) C to 0.8, the probability of faces $\overline{AC}$ and $\overline{BC}$ becomes 0.8, and that of $\overline{AB}$ becomes 0.2. However, in this case, the accumulated opacity of the same ray becomes 0.84, which results in non-zero reconstruction loss. (3) Therefore, with a small weight regularization, we cannot remove C to get this optimal mesh, which contains only $\overline{AB}$, and attains 0 reconstruction loss.	317
F.11	Overview of Reinforce-Ball Algorithm. Based on per-point existence probability ($\Phi(\mathbb{P})$), we sample points for B number of batches ($\mathbf{P}^i$). Here we use $B = 4$, and assume we are reconstructing shape “A”. The points with $\psi = 1$ are rendered in black, while those with $\psi = 0$ are rendered in blue. Then, we identify existing faces in each batch ($\mathbf{F}^i$) based on eq. (7.2). With $\mathbf{P}^i$ and $\mathbf{F}^i$, we compute loss for each batch. Note that the case (1, 2) are better than (3, 4), because they reconstruct the shape better (L_{recon}^i). Also, the case (1) is better than (2), because it has less number of points (L_{card}^i). To minimize the expected loss ($\mathbb{E}[L_{rl}]$), we should maximize the probability to sample the case 1. We optimize $\Phi(\mathbb{P})$ to do that.	319
F.12	Qualitative ablation studies on Reinforce-Ball algorithm (for letter ‘Q’). We render “imaginary” (black) part and “real part” (red, blue) together.	324

Part I

Introduction

Chapter 1: Overview

1.1 Motivation

Modern scientific and engineering problems in the physical world increasingly require solving large-scale optimization and learning tasks. Examples range from controlling autonomous vehicles in complex traffic networks and coordinating robot fleets, to reconstructing detailed 3D geometry for simulation and design. In all of these settings, the underlying phenomena—motion, interaction, and deformation—evolve continuously in time and space. Yet, our computational tools often treat these systems as black boxes or rely on discrete approximations. This mismatch between the continuous nature of the physical world and the discrete nature of our algorithms leads to inefficient exploration, high sample complexity, and suboptimal solutions.

A key observation behind this thesis is that many physical processes are *nearly everywhere differentiable*. When the evolution of a system can be expressed as a differentiable mapping from current states and actions to next states, its derivatives provide powerful information about how to change inputs in order to improve future outcomes. Instead of blindly searching over actions or parameters, we can use gradients as “beacons” that guide optimization. Recent advances in differentiable programming and automatic differentiation have made such gradients accessible for a growing class of simulators and geometric pipelines. However, most existing learning and optimization methods either ignore these gradients, approximate them with surrogate models, or apply them only in small-scale, highly specialized settings.

This disconnect is particularly evident in reinforcement learning (RL) for physical control. Model-free RL methods have achieved impressive results in domains such as video games and recommendation systems, where the environment can be treated as a black box. For continuous control

in physics-based environments, however, naively sampling trajectories and estimating policy gradients from returns is often prohibitively expensive. At the same time, differentiable simulators can provide accurate system-level derivatives of the underlying dynamics, but it remains unclear how to exploit these gradients effectively within standard RL algorithms without introducing excessive variance or bias. Bridging this gap—between gradient-free policy learning and gradient-rich physical models—is a central motivation of the first part of this thesis.

Similar challenges arise in large-scale traffic systems, which form the backbone of modern transportation infrastructure and autonomous driving applications. Traffic dynamics can be modeled at macroscopic scales, using partial differential equations that describe flows of vehicles, or at microscopic scales, using ordinary differential equations that govern the motion of individual drivers. Macroscopic models enable city-level reasoning but can be too coarse for fine-grained control, while microscopic models capture rich local behaviors but are expensive to scale up. Current traffic simulators are typically designed for forward prediction and scenario analysis, not for gradient-based optimization. As a result, tasks such as calibrating model parameters from data, optimizing signal timing, or jointly planning the trajectories of many vehicles often rely on heuristics or gradient-free search. Developing a differentiable, hybrid traffic simulator that retains the strengths of both macro- and micro-scale models, while exposing analytic gradients for control and learning, is therefore a key step toward more scalable “physical AI” in transportation.

A third domain where this tension between continuity and discreteness is pronounced is geometry processing, particularly mesh-based representations of 3D shape. Triangular and tetrahedral meshes remain the de facto standard in graphics, vision, and simulation because they encode explicit topology and are compatible with existing rendering and finite element pipelines. However, meshes are inherently discrete: their connectivity is combinatorial, and traditional meshing algorithms depend

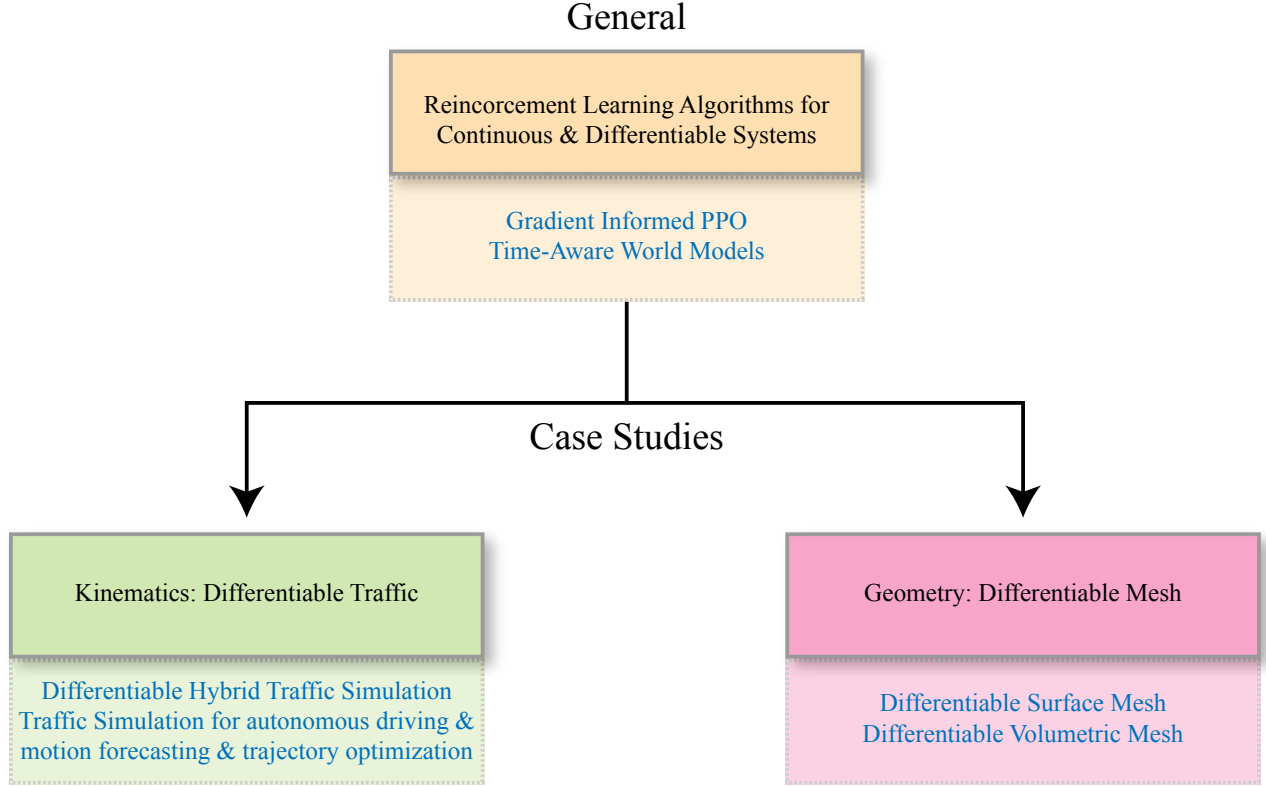


Figure 1.1: The overview of my research. The upper box shows the keyword, and the lower block shows specific work.

on non-differentiable decisions such as edge flips, collapses, and refinement operations. Conversely, many recent learning-based methods adopt differentiable but implicit representations, such as signed distance fields or neural implicit functions, which are well-suited for gradient-based optimization but do not directly yield high-quality meshes suitable for simulation. This creates a methodological gap between differentiable shape optimization and mesh-based downstream tasks.

This thesis is motivated by the belief that these three lines of work—reinforcement learning for physical control, differentiable traffic simulation, and differentiable mesh processing—are all manifestations of the same underlying question:

How can we systematically exploit differentiability in physical and geometric systems to make optimization and learning more efficient, scalable, and physically faithful?

To address this question, my research develops specialized algorithms and frameworks that treat differentiability as a first-class design principle. My overall work can be largely summarized into three categories as shown in Figure 1.1. For each category, I denote its keyword in the upper block and specify my work for it in the lower block. (1) *Develop specialized reinforcement learning algorithms for our physical world*: I will present my work to accelerate the optimization and learning processes for tasks happening in physical environments. With this, I emphasize the importance of differentiable frameworks in problem solving. Then, I will present two case studies of differentiable frameworks in kinematics and geometry, which are core components of our physical world. (2) *Differentiable traffic*: I demonstrate how we can formulate traffic laws in a differentiable manner and use them to solve various problems in the traffic domain. (3) *Differentiable mesh*: Then, I show my differentiable approach to surface and volumetric meshes, which can be used for 3D reconstruction. While pushing the boundaries of these research areas, I have made significant contributions published in major, renowned venues, including SIGGRAPH Asia (TOG), NeurIPS, ICML, ICCV, ICRA, and IROS.

1.2 Technical Challenges

While the promise of differentiability is conceptually simple, turning it into a practical tool for optimization and learning in physical systems raises several technical challenges. In this thesis, I focus on three closely related technical difficulties.

C1. Reliable gradients in the presence of non-smooth and discrete phenomena. Although many physical processes are *nearly everywhere differentiable*, realistic models of the world inevitably contain non-smooth events and discrete structures. In rigid body dynamics, contact, impacts, frictional transitions, and mode switches introduce discontinuities into otherwise smooth dynamics. In traffic,

lane changes, intersection logic, and right-of-way rules behave like discrete state machines layered on top of continuous car-following laws. In geometry processing, triangular and tetrahedral meshes encode combinatorial connectivity, and traditional meshing operations—such as edge flips, collapses, and refinement—are inherently non-differentiable. Naively applying automatic differentiation through such pipelines often yields undefined, unstable, or uninformative gradients exactly where the behavior matters most. The first challenge is therefore to construct formulations that provide *reliable* gradient information even in the presence of non-smooth and discrete phenomena, for example by relaxing discrete decisions into continuous variables, designing differentiable surrogates for topological operations, or carefully localizing non-smooth events while still exposing useful sensitivities to upstream optimizers.

C2. Scalable computation and organization of system-level derivatives. Even when differentiable formulations are available, computing and storing derivatives at the scale required by real-world applications is non-trivial. Backpropagating through long-horizon trajectories or high-resolution fields quickly leads to prohibitive memory usage and runtime. City-scale traffic systems may involve hundreds of thousands or millions of vehicles; volumetric meshes for simulation may contain millions of elements and degrees of freedom. Generic reverse-mode automatic differentiation treats these systems as dense computation graphs, often ignoring problem structure such as local coupling, sparsity, or the separability of independent agents. A second challenge is therefore to design *scalable* differentiable formulations and parallelization schemes that retain exact or controlled-approximation gradients while exploiting domain structure: batching and vectorizing across vehicles in traffic, leveraging sparse operators in mesh-based computations, and using tailored adjoint or Jacobian–vector products that avoid unrolling every intermediate state in memory.

C3. Integrating analytic gradients into stochastic learning algorithms. Finally, even high-quality, efficiently computed system-level gradients do not automatically translate into better learning algorithms. Reinforcement learning methods such as policy gradient and actor–critic are typically derived under the assumption that the environment is a black box, returning only sampled transitions and rewards. Directly injecting analytic derivatives from a differentiable simulator into these updates can introduce bias, numerical sensitivity, or excessive variance, especially in stochastic and long-horizon settings. Similar issues arise when combining analytic gradients with statistical estimates in model-based planning or trajectory optimization. The third challenge is to develop learning rules that *systematically combine* statistical estimators with analytic derivatives, preserving the robustness and generality of standard RL algorithms while benefiting from the additional structure provided by differentiable dynamics.

1.3 Problem Statements

The challenges outlined above can be distilled into a set of concrete research problems that this thesis aims to address. At a high level, the overarching problem can be stated as follows:

High-level Problem. *Given physical and geometric systems that are nearly everywhere differentiable, how can we systematically exploit their differentiable structure to design optimization and learning procedures that are more efficient, scalable, and physically faithful than purely gradient-free or black-box approaches?*

To make this question precise and actionable, I focus on three families of sub-problems corresponding to the main thematic directions of this thesis.

Problem 1: Efficient learning and control in differentiable physical environments. Consider a continuous-control task with state space $\mathcal{S}$, action space $\mathcal{A}$, and dynamics

$$s_{t+1} = f(s_t, a_t), \quad s_t \in \mathcal{S}, \quad a_t \in \mathcal{A},$$

where f is differentiable (almost everywhere) and its gradients $\partial s_{t+1}/\partial s_t$ and $\partial s_{t+1}/\partial a_t$ are available from a differentiable simulator. The objective is to learn a parametrized policy $\pi_\theta(a_t \mid s_t)$ that maximizes expected return $\mathbb{E}[\sum_t \gamma^t r(s_t, a_t)]$. The problem is to design reinforcement learning algorithms that: (i) obtain *reliable* gradient information from the underlying dynamics even in the presence of contacts, non-smooth events, and stochasticity; (ii) *scalably* compute and organize these system-level derivatives along long trajectories; and (iii) *integrate* them with trajectory-based policy gradient estimators in a way that improves sample efficiency and stability over model-free baselines, without introducing unacceptable bias or variance.

Problem 2: Scalable differentiable hybrid traffic simulation for optimization and control. In large-scale traffic systems, the state of the network can be modeled at two coupled levels: a macroscopic description of traffic density and velocity fields over a road graph, and a microscopic description of individual vehicles following car-following laws and lane-change rules. Let $\rho(x, t)$ and $u(x, t)$ denote macroscopic density and velocity fields governed by PDEs, and let $(p_i(t), v_i(t))$ denote the position and velocity of vehicle i governed by ODEs such as the Intelligent Driver Model. The problem is to construct a *hybrid* traffic simulator that: (i) provides a differentiable mapping from initial conditions, control inputs (e.g., signal timings, pace-car policies), and boundary conditions to future macroscopic and microscopic states; (ii) yields *reliable* gradients with respect to these inputs despite

the presence of discrete events such as intersection logic; and (iii) scales to city-level scenarios with hundreds of thousands or millions of vehicles by exploiting parallelism and problem structure. Such a simulator should support downstream tasks including parameter calibration from data, gradient-based signal optimization, and trajectory refinement or forecasting.

Problem 3: Differentiable surface and volumetric mesh generation from observations. In geometry processing, let $\mathcal{O}$ denote observed data such as point clouds, signed distance samples, or multi-view images of a 3D object. The goal is to produce a mesh $M = (V, F)$ or a volumetric tetrahedral mesh $T = (V, K)$ whose geometry and topology are suitable for downstream applications such as rendering or physical simulation. Traditional pipelines solve this via non-differentiable meshing algorithms, while many learning-based methods optimize differentiable implicit representations without directly producing high-quality meshes. The problem is to develop mesh formulations that: (i) encode geometry and connectivity in a *differentiable* manner, providing meaningful gradients with respect to vertex positions and continuous connectivity parameters even when the underlying structures are combinatorial; (ii) admit *scalable* computation of gradients through Delaunay-based constructions, isosurface extraction, or volumetric tessellation; and (iii) enable end-to-end optimization of meshes against observations $\mathcal{O}$, improving both geometric fidelity and element quality over non-differentiable baselines.

Together, these problem statements instantiate the high-level question of how to leverage differentiability in physics and geometry. In the next section, I formalize the central thesis statement and hypotheses that guide the design of the proposed algorithms and systems.

1.4 Thesis Statement and Hypotheses

The problem statements above motivate a unifying perspective on the role of differentiability in physical and geometric systems. Rather than treating gradients as an optional byproduct of numerical methods, this thesis takes the view that differentiability should be a first-class design principle for simulators, representations, and learning algorithms.

Thesis Statement. *By explicitly exploiting differentiable structure in physical dynamics and geometric representations, we can design optimization and learning procedures that are significantly more efficient, scalable, and physically faithful than methods that treat the environment as a black box or rely solely on discrete, non-differentiable pipelines.*

This thesis investigates this claim through three concrete hypotheses, each corresponding to one of the main problem families introduced above.

H1. Gradient-aware reinforcement learning in differentiable environments. *When system-level derivatives are available from a differentiable environment, policy learning algorithms that combine the analytic gradients with statistical estimators can achieve substantial gains in sample efficiency and stability over purely model-free methods, without sacrificing robustness.*

The intuition behind H1 is that analytic gradients of the dynamics provide low-variance directional information about how actions influence future states and rewards, especially in smooth regions of the state space. At the same time, trajectory-based policy gradients are agnostic to modeling errors and can handle non-smooth events and stochasticity. By carefully integrating these two sources of information—rather than replacing one with the other—we can leverage the strengths of both: analytic gradients guide exploration and constrain updates, while stochastic estimators preserve correctness un-

der complex noise and contact patterns. Chapters 2 and 3 test this hypothesis by introducing gradient-informed policy optimization and time-aware world models in differentiable physics environments.

H2. Scalable differentiable hybrid traffic simulation. *By formulating both macroscopic PDE-based traffic models and microscopic ODE-based car-following laws within a unified differentiable framework, and by exploiting their inherent problem structure, we can construct hybrid traffic simulators that scale to city-level scenarios and support gradient-based optimization and control tasks that are impractical for traditional gradient-free approaches.*

The intuition behind H2 is twofold. First, macroscopic and microscopic traffic models each provide complementary advantages: PDEs capture global flow patterns while ODEs capture local driver interactions. A differentiable hybrid formulation allows gradients to propagate across scales, enabling joint optimization of signal timing and vehicle trajectories. Second, the local coupling structure of traffic dynamics—vehicles primarily interact with nearby neighbors and adjacent cells—can be leveraged to design custom parallelization schemes and adjoint computations that avoid the prohibitive costs of naive automatic differentiation. Chapters 4 and 5 investigate this hypothesis by developing a differentiable hybrid traffic model with analytic derivatives and a parallelized simulator for trajectory filtering, reconstruction, and forecasting.

H3. Differentiable mesh representations for surface and volumetric reconstruction. *Relaxing discrete mesh connectivity and volumetric tessellation into differentiable formulations, while carefully regularizing for quality, enables end-to-end optimization of simulation-ready meshes from observations, yielding better trade-offs between geometric fidelity, mesh quality, and element efficiency than traditional non-differentiable pipelines or purely implicit representations.*

The intuition behind H3 is that classical meshing algorithms solve inherently combinatorial problems (e.g., choosing connectivity, flipping edges, refining elements), which are difficult to directly couple with gradient-based optimization. On the other hand, differentiable implicit representations such as signed distance fields provide smooth objectives but do not directly produce high-quality meshes tailored for simulation. By encoding connectivity and volumetric structure into continuous features and differentiable constructions—such as Delaunay-based structures, probabilistic face existence, or differentiable marching tetrahedra—we can optimize geometry and (topologically meaningful) connectivity jointly with respect to observational and regularization losses. Chapters 6–8 examine this hypothesis through differentiable surface mesh representations and volumetric tetrahedral mesh optimization from point clouds and multi-view images.

Together, these hypotheses instantiate the central thesis across different but closely related domains: reinforcement learning for physical control, large-scale traffic simulation, and mesh-based geometry processing. The remainder of this chapter outlines how the subsequent chapters are organized around these themes and how each contributes evidence in support of the thesis.

1.5 Overview of Thematic Approaches

The thesis develops three thematic lines of work that instantiate the above hypotheses in concrete settings: (i) gradient-aware reinforcement learning in differentiable environments, (ii) scalable differentiable hybrid traffic simulation, and (iii) differentiable surface and volumetric mesh representations. Figure 1.1 summarizes these themes at a high level, with each upper block highlighting a core concept and each lower block listing the specific algorithms and systems developed in this dissertation.

Theme A: Gradient-aware reinforcement learning in differentiable environments. The first theme addresses *Problem 1* and *Hypothesis H1* by designing RL algorithms that treat analytic system-level derivatives as a first-class signal. In Chapter 2, I introduce *Gradient Informed Proximal Policy Optimization (GI-PPO)*, which augments PPO with analytic gradients from differentiable environments. GI-PPO constrains policy updates using these gradients, aiming to reduce variance and improve sample efficiency while retaining the robustness of trajectory-based policy gradients. In Chapter 3, I develop a *Time-Aware World Model (TAWM)* that conditions on variable time steps and is trained using a mixture of step sizes. This model leverages continuity of the underlying dynamics to support planning and control across a range of observation rates. Together, these chapters show how analytic derivatives can be safely integrated into stochastic RL algorithms, directly targeting Challenge C1 and Challenge C3 in the context of continuous-control tasks.

Theme B: Scalable differentiable hybrid traffic simulation. The second theme addresses *Problem 2* and *Hypothesis H2* by building differentiable traffic simulators that couple macroscopic and microscopic models and expose analytic derivatives for optimization and control. Chapter 4 presents a *differentiable hybrid traffic model* that combines PDE-based macroscopic dynamics with ODE-based microscopic car-following laws, along with differentiable macro–micro conversion operators. This framework yields analytic gradients with respect to initial conditions and control parameters such as signal timings and pace-car policies. Chapter 5 then focuses on scalability, introducing a *parallelized differentiable traffic simulator* that exploits the local structure of car-following models to simulate and differentiate through trajectories for up to millions of vehicles. These chapters collectively address the need for reliable and scalable gradients (Challenges C1 and C2) in city-scale traffic systems, enabling gradient-based calibration, control, and trajectory optimization.

Theme C: Differentiable mesh representations for surface and volumetric reconstruction. The third theme addresses *Problem 3* and *Hypothesis H3* by developing differentiable formulations for surface and volumetric meshes that can be optimized end-to-end from observations. Chapter 6 introduces *DMesh*, a probabilistic mesh representation in which face existence is encoded by continuous variables derived from weighted Delaunay structures. This formulation allows both geometry and connectivity to be optimized using gradients with respect to point-wise parameters. Chapter 7 presents *DMesh++*, which refines this idea into a more efficient framework with reduced tessellation cost and memory usage, enabling reconstruction of complex shapes from point clouds and multi-view images. Chapter 8 extends the differentiable mesh paradigm to volumetric tetrahedral meshes, proposing a pipeline for generating and optimizing tetrahedral elements via differentiable volumetric marching and mesh-quality-aware loss functions. Across these chapters, the focus is on relaxing discrete connectivity and tessellation decisions into continuous parameters, addressing Challenge C1, and on designing scalable gradient computations suitable for complex meshes, addressing Challenge C2.

Interplay between themes. Although each theme targets a different application domain, they are tightly connected. The RL algorithms in Theme A provide generic tools for exploiting analytic gradients in control problems and are applicable to differentiable traffic simulators and differentiable physics more broadly. The traffic simulators in Theme B demonstrate how large-scale, structured dynamical systems can be made differentiable and used to solve optimization and forecasting tasks that would be impractical with black-box models. The mesh representations in Theme C show how differentiability can be extended from kinematic and traffic dynamics to geometric structures, enabling reconstruction of simulation-ready meshes that could themselves serve as substrates for future differentiable simulation and control. Together, these thematic approaches provide complementary evidence

for the central thesis that differentiability, when explicitly exposed and systematically exploited, can serve as a unifying tool for efficient optimization and learning in physical and geometric systems.

1.6 Related Work

1.6.1 Reinforcement Learning for Physical Tasks

1.6.1.1 Model-Free Algorithms

Model-free Reinforcement Learning (RL) algorithms often use experiences or trajectories from a (black-box) environment and learn policies based on them. As they do not have to learn an additional approximate model of the environment, they often demonstrate rapid convergence to the optimal policy.

Policy Gradient Methods. Policy gradient methods are one of the most popular model-free RL algorithms, and they learn a policy by estimating the gradient of the expected cumulative return with respect to policy parameters [28]. [29] is one of the earliest approaches that use a stochastic estimator to estimate the policy gradient and use it for learning. [30] used the term ‘likelihood-ratio’ (LR) gradient to represent the policy gradient estimated in this way. While being unbiased in nature, policy gradients estimated in this way exhibit high variance, which hampers learning [28]. [31, 32] proposed a way to reduce the high variance of this estimated policy gradient using control variates. [33] proposed an actor-critic method, which is also helpful for reducing the variance. Building on this, [34] presented a more stable advantage estimator as a critic for accelerating training. More recently, [35, 36] were proposed to further stabilize training by constraining policy updates to remain close to the current policy. Now, Proximal Policy Optimization (PPO) [36] is one of the most widely used reinforcement

learning algorithms across various domains, including robotics [37–41] and large language models (LLMs) [42–45], due to its generalizability and robustness.

Analytical Gradient Methods. Instead of a stochastic policy gradient estimator, one can obtain analytic gradients of the policy parameters when the environment is differentiable and thus provides downstream gradients. Analytical gradient methods use these analytically computed policy gradients for learning, but they are still model-free, as they do not train a surrogate model to replicate the environment. [46] proposed one of the earliest approaches to use analytical gradients for learning, which is called Back-Propagation Through Time (BPTT). Compared to policy gradients, analytical gradients exhibit lower variance [47, 48] and therefore contribute to much faster convergence than policy gradients. However, [49–51] point out that analytical gradient methods can suffer from the exploding or vanishing gradients problem when the horizon becomes longer, due to the chaotic nature of environments. Furthermore, [52] reports that these methods are vulnerable to the empirical bias problem in learning. To amend these issues, [14] propose a truncated window approach, and [50, 52] claim to use a mixture of policy gradients and analytical gradients. With these corrective measures, analytical gradients are often used in solving physical tasks that have a high-dimensional search space [14, 49, 53–63].

1.6.1.2 Model-based Algorithms

Despite its generalizability and robustness, the lack of sample efficiency is often referred to as a main weakness of model-free RL algorithms. Model-based RL (MBRL) algorithms tackle this sample-efficiency problem by training a surrogate model that approximates the given environment during training. This surrogate model lets the agent predict what the outcome of the agent’s action would be without accessing the true environment [64]. Because the actor can obtain an unlimited number

of simulated samples from the surrogate model, it is inherently more sample efficient than model-free methods. While early attempts like [50, 65] used a Gaussian Process (GP) to define this surrogate model, recently neural networks are preferred over them because of their superior representational power [66–71]. Specifically, Dreamer and its variants [67–69, 72] train policies directly within the learned model. On the other hand, model-predictive-control (MPC) methods [70, 71] rely on action planning. These MBRL methods are also widely used to solve physical tasks, mainly due to their generalizability to multi-task environments and scalability [72–77].

1.6.2 Differentiable Kinematics

1.6.2.1 Differentiable Physics Simulation

Physics simulators have traditionally been used for solving optimization problems in various domains, such as topology design [78, 79], aerodynamics/fluid dynamics [80, 81], and meteorology [82, 83]. When it comes to physics simulators for simulating rigid bodies, soft bodies, or fluids, differentiable simulators have gained popularity as they provide system-level gradients, which can be used for advanced data sampling for efficient optimization and learning [84]. Specifically, [55, 57] proposed differentiable rigid-body physics simulators that could be used for robotics learning. [59] proposed a differentiable MPM simulator for soft bodies, and [85] presented a differentiable cloth simulator. [53] went one step further by coupling rigid-body and cloth simulation. When it comes to projective dynamics, [86, 87] presented differentiable projective dynamics-based simulators for simulating multiple soft bodies. [88, 89] proposed differentiable fluid simulation, or its coupling with rigid bodies, for optimization. In parallel to these works, various software tools were developed to help implement differentiable simulations [58, 90, 91] – these frameworks often support automatic differ-

entiation. Based on these research efforts, the standard physics simulators that people use in practice, such as MuJoCo [92], Isaac Sim [93], Genesis [94], and Brax [49], are also trying to extend their support for differentiable physics to accelerate optimization and learning. They have become especially important nowadays, as people are trying to use physics simulators for generating synthetic datasets for training “world models” that understand real-world physical laws, and gradients from these simulators play a vital role in this application.

1.6.2.2 Traffic Simulation

Based on the physical principles that describe traffic flow, traffic simulations are used to solve various optimization and learning problems in the traffic domain.

Traffic Models. Techniques for traffic simulation can be broadly classified as either macroscopic or microscopic based on their modeling assumptions. In macroscopic formulations, traffic flow is described using density and flux. [95, 96] suggested one of the earliest models, called the LWR model, based on this idea, which is essentially a nonlinear PDE on the density of vehicles. [97, 98] added a momentum term to the LWR model, which allowed them to model more complex traffic dynamics. [99, 100] modified the momentum term in the previous model based on the observation that traffic flow is anisotropic, which is called the ARZ model. [101, 102] used this ARZ model to visualize traffic flow and solve traffic congestion problems.

On the other hand, microscopic models describe the motion of individual vehicles, typically tracking velocity, the bumper-to-bumper distance to the leading vehicle, and the relative velocity between them [103]. [104–106] are regarded as the initial attempts for this microscopic approach. [107–110] proposed modern variations of these models, including behavioral traits of individual ve-

hicles, which make simulations more realistic and descriptive. Among them, the Intelligent Driver Model (IDM) [109] is one of the most widely adopted microscopic models in recent simulators.

In general, macroscopic models are computationally more efficient than microscopic ones when simulating a large number of vehicles, as their computational cost is proportional to the length of the simulated road, not the number of vehicles therein. However, microscopic models provide more fine-grained details of vehicle motion than macroscopic ones, because every individual vehicle observes its surrounding environment and decides its actions. [111–114] try to find a middle ground between these two regimes; these hybrid models often simulate only the regions of interest with microscopic models for accuracy and use macroscopic models elsewhere to maintain overall flow correctness while attaining computational efficiency.

Traffic Simulators. Modern traffic simulators like SUMO [115], HighwayEnv [116], CARLA [117], and FLOW [118] use the Intelligent Driver Model (IDM) to model vehicle movements. On the other hand, simulators like MATSIM [119] and VISSIM [120] employ similar car-following models, but not IDM. Despite their widespread usage in traffic problem solving, these simulators have a limitation in computational efficiency because they are implemented in a sequential way. More recently, advancements in machine learning and GPU hardware have led to the development of parallelized or differentiable traffic simulators for large-scale traffic simulation and urban mobility optimization. [121] proposed a differentiable formulation of IDM, but it does not support parallelization. In contrast, non-differentiable MOSS [122] uses GPU power for city-scale traffic simulation based on IDM and MOBIL [123]. Data-driven simulators like Waymax [124] and GPUDrive [125] offer both GPU acceleration and differentiation, but focus on replaying real-world trajectory logs from the Waymo Open Dataset (WOMD) [126], without reactive traffic behaviors.

1.6.3 Differentiable Geometry

1.6.3.1 Neural Implicit Shape Representations

Recently, the use of neural implicit functions for shape representation has gained popularity in graphics and vision applications [5, 127–132], due to their innate differentiability and the development of neural networks. They mainly use a neural network to encode density and additional visual features of a space. [5, 6, 133–137] proposed using neural signed distance functions (SDFs) to enhance geometric accuracy. [7, 8] are extensions of these methods, which use neural unsigned distance fields (UDFs) to represent thin shells instead of volumes. After optimization is done, they extract the explicit geometry using iso-surface extraction techniques [22, 23, 138]. Nowadays, these neural implicit shape representations are widely used in 3D reconstruction [5, 6, 127, 132, 134] and 3D generation [139–145].

1.6.3.2 Explicit Mesh Representation

While neural implicit representations have gained popularity in machine learning, explicit surface or volumetric mesh representations are still favored in downstream applications, such as rendering, geometric design, or physics simulation. For instance, volumetric tetrahedral meshes are widely used in volumetric rendering [146–150], fluid simulation [151–153], soft-body simulation [154–157], and fracture simulation [158]. However, adopting meshes into current machine learning frameworks has not been as easy as neural implicit representations, because the connectivity information of the mesh is discrete in nature. Therefore, some workarounds have been developed to handle meshes in current machine learning or gradient-based optimization pipelines.

Differentiable Mesh with Fixed Topology. As it is difficult to model changing connectivity during mesh optimization, some approaches freeze the connectivity and only optimize vertex positions [3, 4, 24, 159–161]. To improve mesh quality, they often adopt remeshing during optimization, which is not differentiable.

Differentiable Mesh Extraction from Implicit Fields. There are other methods that represent a shape with an implicit field using vertex-wise signed distance function (SDF) values and extract a surface or volumetric mesh using differentiable isosurface extraction algorithms. [9, 162] proposed a deformable background tetrahedral grid to encode the SDF information and extract the surface mesh using the Marching Tetrahedra (MT) [23] algorithm. [11] presents a more advanced version, which uses a background cubic grid to encode the SDF and utilizes a variant of the Dual Contouring (DC) algorithm to extract a surface or volumetric mesh. More recently, [12] proposed using two layers of SDF to extract thin shells instead of closed surfaces from the implicit field. Due to their differentiability and ability to represent diverse topologies, these methods are widely used in 3D reconstruction [10, 163] and generation [164–166], similarly to neural implicit methods.

1.7 Outline and Contributions

This dissertation is organized into five parts.

Part I – Introduction. Chapter 1 provides an overview of the thesis. It motivates the need for exploiting differentiability in physical and geometric systems, formalizes the core research questions and hypotheses, and outlines the three thematic threads: gradient-informed reinforcement learning for physical control, differentiable traffic simulation, and differentiable mesh processing for shape

reconstruction.

Part II – Reinforcement Learning for Physical Tasks. In Part II, I develop reinforcement learning algorithms that explicitly leverage system-level derivatives in differentiable environments.

Chapter 2 introduces Gradient-Informed Proximal Policy Optimization (GI-PPO), a policy learning method that augments PPO with analytical gradients from differentiable simulators. The chapter proposes the notion of an α -policy as a locally superior policy, derives criteria to diagnose variance and bias in analytical gradients, and shows how to adaptively modulate their influence during learning. Extensive experiments on function optimization, differentiable physics, and traffic control demonstrate accelerated convergence compared to gradient-free baselines.

Chapter 3 presents the Time-Aware World Model (TAWM), a continuous-time model-based RL approach that conditions on the time-step size Δt and is trained on a mixture of time steps. By learning a single latent dynamics model that is consistent across multiple temporal resolutions, TAWM achieves improved sample efficiency and robustness on control tasks and PDE control problems, while providing theoretical insight into the benefits of multi-scale temporal supervision.

Part III – Differentiable Kinematics: Traffic. Part III focuses on differentiable kinematic models for large-scale traffic systems.

Chapter 4 develops a differentiable hybrid traffic simulator that tightly couples a macroscopic PDE-based flow model with a microscopic ODE-based car-following model. The chapter formulates both scales in a unified differentiable framework, derives analytic derivatives for macroscopic evolution, microscopic dynamics, and macro–micro conversions, and demonstrates how these gradients can be used for parameter calibration, control, and policy learning in realistic traffic networks.

Chapter 5 builds on this simulator to propose a parallelized differentiable traffic engine for gradient-based trajectory optimization. It introduces a batched simulation scheme that scales to many vehicles and scenarios in parallel, analyzes and mitigates unrealistic behaviors induced by naïve gradient descent, and applies the resulting system to trajectory filtering, reconstruction, and forecasting. The chapter highlights how exact kinematic gradients enable efficient, training-free baselines and serve as a foundation for learned predictors.

Part IV – Differentiable Geometry: Mesh. Part IV investigates differentiable mesh representations for surface and volumetric geometry.

Chapter 6 introduces DMesh, a probabilistic surface-mesh representation that relaxes triangle existence into continuous variables over a weighted Delaunay triangulation. By formulating mesh connectivity as soft probabilities and defining differentiable losses on geometry, regularity, and data fidelity, DMesh enables gradient-based optimization over both connectivity and vertex positions, and demonstrates applications to point-cloud reconstruction and multi-view reconstruction.

Chapter 7 presents DMesh++, an efficient extension of DMesh tailored to complex, high-resolution shapes. It decouples connectivity selection from geometric optimization using a minimum-ball construction, reduces the computational cost of probability evaluation and tessellation, and improves reconstruction quality on challenging shapes while maintaining differentiability.

Chapter 8 develops a differentiable pipeline for volumetric meshing based on Delaunay tetrahedralization and volumetric Marching Tetrahedra. The chapter defines a loss-driven procedure that optimizes volumetric meshes with respect to signed distance fields, multi-view image cues, and mesh-quality metrics, enabling gradient-based control of internal element quality and demonstrating compatibility with downstream physics simulation.

Part V – Conclusion. Chapter 9 summarizes the main findings of the thesis, reflects on the unifying role of differentiability across reinforcement learning, traffic simulation, and geometric modeling, and outlines several directions for future work in scalable differentiable systems and physical AI.

Summary of Contributions. The main technical contributions of this dissertation are:

- **Gradient-informed policy optimization in differentiable environments (Chapter 2).** I propose GI-PPO, a variant of PPO that integrates analytical gradients from differentiable simulators via an α -policy formulation, together with diagnostic metrics for variance and bias that allow the algorithm to selectively trust or downweight simulator gradients, leading to faster and more stable policy improvement.
- **Time-Aware World Model for multi-scale continuous-time dynamics (Chapter 3).** I introduce TAWM, a continuous-time world model trained on a mixture of time-step sizes, showing both theoretically and empirically that conditioning on Δt enables a single model to capture high- and low-frequency dynamics, thereby improving sample efficiency and robustness in model-based RL and PDE control.
- **Differentiable hybrid macro–micro traffic simulator with analytic gradients (Chapter 4).** I design a unified differentiable traffic framework that combines macroscopic flow models and microscopic car-following dynamics, derive closed-form gradients for each component and their conversions, and demonstrate how these gradients can be used for parameter calibration, signal optimization, and policy learning in realistic traffic networks.
- **Parallelized differentiable traffic simulation for trajectory optimization and forecasting (Chapter 5).** I develop a GPU-accelerated, batched differentiable traffic simulator that supports

large-scale gradient-based trajectory optimization, introduce strategies to detect and suppress unrealistic gradient-driven behaviors, and show applications to trajectory filtering, reconstruction, and prediction as efficient baselines for learned forecasting models.

- **Probabilistic differentiable surface-mesh representation (Chapter 6).** I propose DMesh, a differentiable mesh formulation that relaxes triangle connectivity into continuous probabilities over a weighted Delaunay triangulation, enabling end-to-end gradient-based optimization of mesh connectivity and geometry from point clouds and multi-view images while maintaining compatibility with explicit mesh pipelines.
- **Efficient differentiable mesh modeling for complex shapes (Chapter 7).** I extend DMesh to DMesh++, which leverages a minimum-ball based construction to reduce the computational overhead of probability evaluation and tessellation, improving scalability to complex, high-resolution shapes and yielding higher-quality reconstructions under much lower computational cost.
- **Differentiable Delaunay tetrahedral mesh generation for volumetric reconstruction (Chapter 8).** I introduce a differentiable volumetric meshing pipeline that combines Delaunay tetrahedralization with volumetric Marching Tetrahedra and mesh-quality losses, enabling gradient-based optimization of tetrahedral meshes with respect to signed distance fields, multi-view observations, and FEM-oriented quality metrics, and validating the resulting meshes in downstream physics simulations.

Part II

Reinforcement Learning for Physical Tasks

Chapter 2: Gradient Informed Proximal Policy Optimization

This chapter presents the first concrete instantiation of the thesis’s central idea: leveraging gradient information to guide efficient optimization in high-dimensional decision-making problems. We introduce a policy learning framework that integrates analytical gradients from differentiable environments into Proximal Policy Optimization (PPO), using the notion of an α -policy to regulate the influence of these gradients during learning. This formulation demonstrates how gradient-informed reasoning can systematically improve exploration and convergence, even in environments that exhibit complex interactions and partial discontinuities.

2.1 Introduction

Reinforcement learning (RL) techniques often take gradient-free experiences or trajectories to collect information and learn policies—suitable for many applications such as video games and recommender systems. In contrast, differentiable programming offers analytical gradients that describe the action-result relationship (Equation 2.2). In particular, recent developments in programming tools [167] have enabled direct access to accurate analytical gradients of the environmental dynamics, as exemplified in physics simulations [49, 53, 55–57], instead of using differentiable models for approximation [64, 65]. Policy gradients estimated from the integration of these analytical gradients and the reparameterization trick (RP gradient) [47] often display less variance than likelihood ratio policy gradients (LR gradient) [29, 30], derived using the log-derivative trick. Consequently, a subset of research has explored the utilization of these analytical gradients for RL applications, especially in physics domains [14, 53, 54].

Nevertheless, in environments with inherent chaotic nature, the RP gradient is found to exhibit high variance [50] and even empirical bias [52]. Therefore, a strategic approach is necessary for the adoption of analytical gradients in policy learning [51]. One such strategy involves the interpolation of the LR and RP gradients based on their variance and bias [50, 52], made possible by the fact that both are different estimators of the same policy gradient. However, this approach requires estimating the LR and RP gradient for each sample trajectory to compute sample variance, a process that can be time-consuming (Appendix A.7). Additionally, for on-policy RL methods that do not initially estimate the LR gradient, such as Trust Region Policy Optimization (TRPO) [35] or Proximal Policy Optimization (PPO) [36] algorithms, it remains uncertain how to evaluate the variance and bias of analytical gradients. These gradients, however, continue to be desirable when they exhibit low variance and provide insightful information about environmental dynamics.

Given these challenges, our study explores how to incorporate analytical gradients into the PPO framework, taking their variance and bias into account. Our primary contributions are threefold: (1) we propose a novel method to incorporate analytical gradients into the PPO framework *without the necessity of estimating LR gradients*, (2) we introduce an adaptive α -policy that allows us to dynamically adjust the influence of analytical gradients by evaluating the variance and bias of these gradients, and (3) we show that our proposed method, GI-PPO, strikes a balance between analytical gradient-based policy updates and PPO-based updates, yielding superior results compared to baseline algorithms in various environments.

We validate our approach across diverse environments, including classical numerical optimization problems, differentiable physics simulations, and traffic control environments. Among these, traffic environments typify complex real-world scenarios where analytical gradients are biased. We demonstrate that our method surpasses baseline algorithms, even under these challenging conditions.

2.2 Related Work

2.2.1 Policy Learning Without Analytical Gradients

The policy gradient is described as the gradient of the expected cumulative return in relation to policy parameters [168]. For a stochastic policy, as examined in this paper, REINFORCE [169] represents one of the initial methods to employ a statistical estimator for the policy gradient to facilitate policy learning. The policy gradient obtained with this estimator is commonly referred to as a likelihood-ratio (LR) gradient [30, 170]. Despite the LR gradient’s ability to operate without analytical gradients of dynamics and its unbiased nature, it frequently exhibits high variance [28]. Furthermore, the nonstationarity of the policy renders stable learning difficult to guarantee [31, 34].

Several strategies have been proposed to address the high variance issue inherent to the LR gradient, including the use of additive control variates [31, 32]. Among these, the actor-critic method [33] is highly favored. However, the introduction of a value function may induce bias. To counteract this, the Generalized Advantage Estimator (GAE) [34] was developed to derive a more dependable advantage function. Concerning the issue of unstable learning, methods such as TRPO [35] and PPO [36] propose optimizing a surrogate loss function that approximates a random policy’s expected return. By constraining the policy updates to remain close to the current policy, these methods significantly enhance stability in learning compared to pure policy gradient methods, which makes them more preferred nowadays.

2.2.2 Policy Learning With Analytical Gradients

In contrast to the LR gradient, the RP gradient is based on the reparameterization trick, which mandates analytical gradients of dynamics yet typically achieves lower variance [47, 48]. The RP gradient can be directly estimated over the entire time horizon using Backpropagation Through Time (BPTT) [46] and subsequently used for policy updates. However, BPTT has been found to encounter multiple optimization challenges such as explosion or vanishing of analytical gradients over extended trajectories [49]. This is because, surprisingly, the RP gradient, contrary to earlier beliefs, may experience exploding variance due to the chaotic nature of environments [50, 51]. Additionally, [52] noted that the RP gradient could also be subject to empirical bias.

Several corrective measures exist for the issues concerning the RP gradient [51], such as using a truncated time window bootstrapped with a value function [14]. This approach has demonstrated its efficacy in addressing physics-related problems. Another class of solutions involves interpolating between LR and RP gradients by evaluating their sample variance and bias [50, 52]. Likewise, when we carefully address the problems of RP gradient, it can provide valuable information helpful for policy learning. However, when it comes to PPO, which is one of the most widely used RL methods nowadays, it is unclear how we can leverage the analytical gradients that comprise the RP gradient, as it does not estimate LR gradient which can be used for comparison to the RP gradient. In this paper, we address this issue and show that we can use analytical gradients to enhance the performance of PPO.

2.3 Preliminaries

2.3.1 Goal

We formulate our problems as a Markov decision process (MDP) defined by a tuple $(S, A, P, r, \rho_0, \gamma)$, where S is a set of states, A is a set of actions, $P : S \times A \times S \rightarrow \mathbb{R}$ is a state transition model, $r : S \times A \rightarrow \mathbb{R}$ is a reward model, ρ_0 is the probability distribution of the initial states, and γ is a discount factor.

Under this MDP model, our goal is to train a parameterized stochastic policy $\pi_\theta : S \times A \rightarrow \mathbb{R}^+$, where θ represents parameters, that maximizes its expected sum of discounted rewards, $\eta(\pi_\theta)$:

$$\eta(\pi_\theta) = \mathbb{E}_{s_0, a_0, \dots \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \quad (2.1)$$

In this paper, we denote the current policy that is used to collect experience as $\pi_{\bar{\theta}}$.

2.3.2 Analytical Gradients

We assume that S and A are continuous, differentiable spaces, and P and r are also differentiable models. Then, our differentiable environment provides us with following analytical gradients of observation or reward at a later timestep with respect to the action at the previous timestep:

$$\frac{\partial s_{t+1+k}}{\partial a_t}, \frac{\partial r_{t+k}}{\partial a_t}, \quad (2.2)$$

where $k \geq 0$ is an integer. With these basic analytical gradients, we can compute the gradient of certain advantage function, which we denote as $\frac{\partial A}{\partial a}$. In this paper, we use Generalized Advantage

Estimator (GAE) [34]. See Appendix A.2.1 for details.

2.3.3 Policy Update

In this section, we discuss how we update policy in two different settings: the policy gradient method based on RP gradient and PPO.

2.3.3.1 RP Gradient

To compute the RP gradient, we rely on the reparameterization trick [47]. This trick requires us that we can sample an action a from π_θ by sampling a random variable ϵ from some other independent distribution q . To that end, we assume that we use a continuous differentiable bijective function $g_\theta(s, \cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^n$ that maps ϵ to an action a , where n is the action dimension. Then the following holds because of injectivity,

$$\left| \det\left(\frac{\partial g_\theta(s, \epsilon)}{\partial \epsilon}\right) \right| > 0, \forall \epsilon \in \mathbb{R}^n, \quad (2.3)$$

and we can define the following relationship.

Definition 2.3.1 We define $\pi_\theta \triangleq g_\theta$ if following holds for an arbitrary open set $T_\epsilon \subset \mathbb{R}^n$:

$$\int_{T_a} \pi_\theta(s, a) da = \int_{T_\epsilon} q(\epsilon) d\epsilon,$$

where $T_a = g_\theta(s, T_\epsilon)$.

Lemma 2.3.2 If $\pi_\theta \triangleq g_\theta$,

$$\pi_\theta(s, a) = q(\epsilon) \cdot \left| \det\left(\frac{\partial g_\theta(s, \epsilon)}{\partial \epsilon}\right) \right|^{-1},$$

where $a = g_\theta(s, \epsilon)$. The inverse is also true.

Proof: See Appendix [A.1.1](#).

Specifically, we use $q = \mathcal{N}(0, I)$ in this paper, and define g_θ as follows,

$$g_\theta(s, \epsilon) = \mu_\theta(s) + \sigma_\theta(s) \cdot \epsilon \quad (\|\sigma_\theta(s)\|_2 > 0),$$

which satisfies our assumption.

With $g_\theta(s, \epsilon)$ and the analytical gradients from Equation [2.2](#), we obtain RP gradient by directly differentiating the objective function in Equation [2.1](#), which we use for gradient ascent. See Appendix [A.2.2](#) for details.

2.3.3.2 PPO

PPO relies on the observation that we can evaluate a policy π_θ with our current policy $\pi_{\bar{\theta}}$, using its advantage function $A_{\pi_{\bar{\theta}}}$ as

$$\eta(\pi_\theta) = \eta(\pi_{\bar{\theta}}) + \int_s \rho_{\pi_\theta}(s) \int_a \pi_\theta(s, a) A_{\pi_{\bar{\theta}}}(s, a),$$

where $\rho_{\pi_\theta}(s)$ is the discounted visitation frequencies. See [[35](#), [171](#)] for the details.

As far as the difference between $\pi_{\bar{\theta}}$ and π_θ is sufficiently small, we can approximate $\rho_{\pi_\theta}(s)$ with $\rho_{\pi_{\bar{\theta}}}(s)$ [[35](#)], and get following surrogate loss function:

$$L_{\pi_{\bar{\theta}}}(\pi_\theta) = \eta(\pi_{\bar{\theta}}) + \int_s \rho_{\pi_{\bar{\theta}}}(s) \int_a \pi_\theta(s, a) A_{\pi_{\bar{\theta}}}(s, a). \quad (2.4)$$

Note that we can estimate this loss function with Monte Carlo sampling as described in Ap-

pendix A.2.3. Since this is a local approximation, the difference between $\pi_{\bar{\theta}}$ and π_{θ} must be small enough to get an accurate estimate. TRPO [35] and PPO [36] constrain the difference to be smaller than a certain threshold, and maximize the right term of (2.4). In particular, PPO restricts the ratio of probabilities $\pi_{\theta}(s_i, a_i)$ and $\pi_{\bar{\theta}}(s_i, a_i)$ for every state-action pair (s_i, a_i) in the buffer as follows to attain the goal, using a constant ϵ_{clip} :

$$1 - \epsilon_{clip} < \frac{\pi_{\theta}(s_i, a_i)}{\pi_{\bar{\theta}}(s_i, a_i)} < 1 + \epsilon_{clip}. \quad (2.5)$$

2.4 Approach

In this section, we discuss how we can use analytical gradients in the PPO framework while considering their variance and biases. We start our discussion with the definition of α -policy.

2.4.1 α -Policy

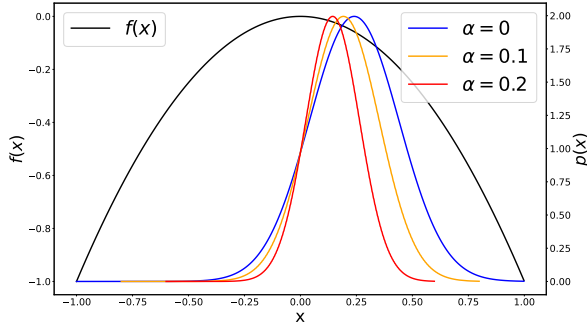
With the analytical gradient of advantage with respect to an action ($\nabla_a A_{\pi_{\bar{\theta}}}(s, a)$) from our differentiable environments, we can define a new class of policies π_{α} , parameterized by α , as follows.

Definition 2.4.1 *Given current policy $\pi_{\bar{\theta}}$, we define its α -policy as follows:*

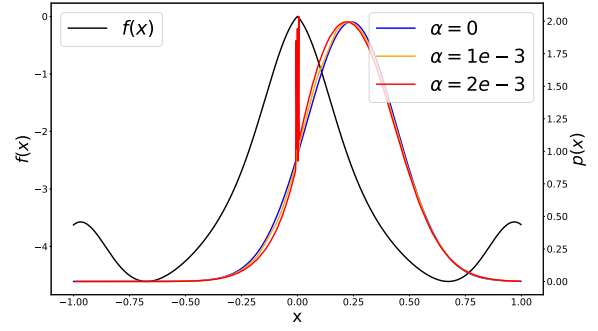
$$\pi_{\alpha}(s, \tilde{a}) = \begin{cases} \frac{\pi_{\bar{\theta}}(s, a)}{|\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))|} & \text{if } \exists a \text{ s.t. } \tilde{a} = f(a) = a + \alpha \cdot \nabla_a A_{\pi_{\bar{\theta}}}(s, a) \\ 0 & \text{else} \end{cases}, \quad (2.6)$$

$$\text{where constant } |\alpha| < \frac{1}{\max_{(s, a)} |\lambda_1(s, a)|}.$$

Here $\lambda_1(s, a)$ represents the minimum eigenvalue of $\nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a)$.



(a) De Jong's Function



(b) Ackley's Function

Figure 2.1: α -policies for 1-dimensional (a) De Jong's function and (b) Ackley's function. The original policy is rendered in blue, and alpha policies for different α values are rendered in orange and red. Those α -policies are obtained from analytical gradients of the given function $f(x)$ rendered in black. Note that Ackley's α -policy becomes invalid when $\alpha = 2 \cdot 10^{-3}$, because of singularities near 0.

Lemma 2.4.2 *Mapping f is injective, and for an arbitrary open set of action $T \subset A$, $\pi_\alpha(s, \cdot)$ selects a set of action $\tilde{T} = f(A)$ with the same probability that $\pi_{\bar{\theta}}(s, \cdot)$ selects A .*

Proof: See Appendix A.1.2.

In Figure 2.1, we provide renderings of α -policies for different α s when we use 1-dimensional De Jong's function and Ackley's function [172] as our target functions. The illustration provides us an intuition that when α is sufficiently small, π_α can be seen as a policy that selects slightly better action than $\pi_{\bar{\theta}}$ with the same probability. In fact, we can prove that π_α indeed gives us better estimated expected return in Equation 2.4 when $\alpha > 0$ is sufficiently small.

Proposition 2.4.3 *If $|\alpha| \ll 1$,*

$$\begin{aligned} L_{\pi_{\bar{\theta}}}(\pi_\alpha) - \eta(\pi_{\bar{\theta}}) &= O(|\alpha|) \text{ when } \alpha > 0, \\ \eta(\pi_{\bar{\theta}}) - L_{\pi_{\bar{\theta}}}(\pi_\alpha) &= O(|\alpha|) \text{ when } \alpha < 0, \end{aligned} \tag{2.7}$$

where $L_{\pi_{\bar{\theta}}}$ denotes estimated expected return defined in Equation 2.4.

Proof: See Appendix [A.1.3](#).

This proposition tells us that α -policy fits into PPO framework seamlessly, and thus if we update our policy towards α -policy, it does not contradict the PPO's objective. However, since there is a second order derivative $\nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a)$ in the definition, it is unclear how we can update our policy π_{θ} towards π_{α} . In the next section, we show how we can do this and how it is related to RP gradient.

2.4.2 Relationship between RP gradient and α -policy

To show how we can approximate α -policy, for a deterministic function $g_{\bar{\theta}}$ such that $\pi_{\bar{\theta}} \triangleq g_{\bar{\theta}}$, we define a new function g_{α} as follows, and provide a Lemma about it.

$$g_{\alpha}(s, \epsilon) = a + \alpha \cdot \nabla_a A_{\pi_{\bar{\theta}}}(s, a), \text{ where } a = g_{\bar{\theta}}(s, \epsilon). \quad (2.8)$$

Lemma 2.4.4 When $a = g_{\bar{\theta}}(s, \epsilon)$,

$$\det\left(\frac{\partial g_{\alpha}(s, \epsilon)}{\partial \epsilon}\right) \cdot \det\left(\frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon}\right)^{-1} = \det(I + \alpha \cdot \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a)). \quad (2.9)$$

Proof: See Appendix [A.1.4](#).

Note that we can update g_{θ} to approximate g_{α} by minimizing the following loss.

$$L(\theta) = \mathbb{E}_{s_0, \epsilon_0, \dots \sim q} [||g_{\theta}(s_t, \epsilon_t) - g_{\alpha}(s_t, \epsilon_t)||^2]. \quad (2.10)$$

In fact, it turns out that minimizing Equation [2.10](#) leads our policy π_{θ} to approximate π_{α} , because of the following Proposition.

Proposition 2.4.5 *If $\pi_{\bar{\theta}} \triangleq g_{\bar{\theta}}$, for α that satisfies the constraint in Definition 2.4.1, $\pi_{\alpha} \triangleq g_{\alpha}$.*

Proof: See Appendix A.1.6.

Note that we can use different advantage function formulations for A in Equation 2.8, such as GAE [34] that we use in this work. However, let us we consider the following advantage function $\hat{A}$ to see the relationship between RP gradient and α -policy.

$$\hat{A}_{\pi_{\theta}}(s_t, a_t) = \frac{1}{2} \mathbb{E}_{s_t, a_t, \dots \sim \pi_{\theta}} \left[\sum_{k=t}^{\infty} \gamma^k r(s_k, a_k) \right], \quad (2.11)$$

Then, the following Lemma holds.

Lemma 2.4.6 *When $\alpha = 1$, if we define g_{α} with $\hat{A}$, the RP gradient corresponds to $\frac{\partial L}{\partial \theta}$ at $\theta = \bar{\theta}$.*

Proof: See Appendix A.1.5.

This Lemma tells us that we can understand the RP gradient as the very first gradient we get when we update our policy toward π_{α} by minimizing Equation 2.10. In other words, α -policy is not only a superior policy than the original one in the PPO’s point of view (Proposition 2.4.3), but also a target policy for RP gradient (Lemma 2.4.6). Therefore, we conclude that α -policy is a bridge that connects these two different policy update regimes.

2.4.3 Algorithm

Our algorithm is mainly comprised of 3 parts: (1) Update π_{θ} to π_{α} by minimizing Equation 2.10, (2) Adjust α for next iteration, and (3) Update π_{θ} by maximizing Equation 2.4. In this section, we discuss details about Step 2 and 3, and provide the outline of our algorithm.

2.4.3.1 Step 2: Adjusting α

In Definition 2.4.1, we can observe that α controls the influence of analytical gradients in policy updates. If $\alpha = 0$, it is trivial that $\pi_\alpha = \pi_{\bar{\theta}}$, which means that analytical gradients are ignored. In contrast, as α gets bigger, analytical gradients play bigger roles, and π_α becomes increasingly different from $\pi_{\bar{\theta}}$. Then, our mission is to find a well-balanced α value. Rather than setting it as a fixed hyperparameter, we opt to adjust it adaptively. We first present the following variance and bias criteria to control α , as it is commonly used to assess the quality of gradients [50, 52].

Variance After we approximate π_α in Step 1), we can estimate $\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))$ for every state-action pair in our buffer using Lemma 2.4.4, and adjust α to bound the estimated values in a certain range, $[1 - \delta, 1 + \delta]$. There are two reasons behind this strategy.

- As shown in Definition 2.4.1, we have to keep α small enough so that π_α does not become invalid policy. By keeping $\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))$ larger than $1 - \delta$ to keep its positiveness, we guarantee that our policy is not updated towards an invalid policy.
- Note that $\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a)) = \prod_{i=1}^n (1 + \alpha \lambda_i)$, where λ_i is the i -th eigenvalue of $\nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a)$. By constraining this value near 1 by adjusting α , we can guarantee stable policy updates even when some eigenvalues of $\nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a)$ have huge magnitudes, which is related to a high variance RP gradient. In Appendix A.3.1, we provide empirical evidence that this estimation is related to the sample variance of analytical gradients.

Bias By Proposition 2.4.3, we already know that when we estimate the right term of Equation 2.4 after Step 1), it should be a positive value. However, if the analytical gradients were biased, this

condition could not be met. In runtime, if we detect such cases, we decrease α .

In addition, we use one more criterion to adjust α , which we call the out-of-range ratio. This criterion was designed to promise the minimum amount of policy update by PPO.

Out-of-range-ratio We define out-of-range-ratio as follows,

$$\text{out-of-range-ratio} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}\left(\left|\frac{\pi_{\theta}(s_i, a_i)}{\pi_{\hat{\theta}}(s_i, a_i)} - 1\right| > \epsilon_{clip}\right), \quad (2.12)$$

where N is the size of the buffer, $\mathbb{I}$ is the indicator function, and ϵ_{clip} is the constant from Equation 2.5.

We assess this ratio after the policy update in Step 1), allowing its value to be non-zero when $\alpha > 0$.

A non-negligible value of this ratio indicates a potential violation of the PPO’s restriction outlined in Equation 2.5, thereby compromising the effectiveness of PPO. Further elaboration can be found in Appendix A.3.2. Therefore, when this ratio exceeds a predetermined threshold, we subsequently reduce the value of α .

To sum it up, our strategy to control α aims at decreasing it when analytical gradients exhibit high variance or bias, or when there is little room for PPO updates. This results in greater dependence on PPO, which can be thought of as a safeguard in our approach. Otherwise, we increase α , since we can expect higher expected return with a greater α as shown in Proposition 2.4.3.

2.4.3.2 Step 3: PPO Update

Since we do policy updates based on PPO after the update based on π_{α} , there is a possibility that the PPO update can “revert” the update from Step 1). To avoid such situations, we propose to use the

following π_h in the place of $\pi_{\bar{\theta}}$ in Equation 2.5:

$$\pi_h(s, a) = \frac{1}{2}(\pi_{\bar{\theta}}(s, a) + \pi_{\alpha}(s, a)).$$

By using this virtual policy π_h , we can restrict the PPO update to be done near both $\pi_{\bar{\theta}}$ and π_{α} . See Appendix A.3.2 for more details.

2.4.3.3 Pseudocode

In Algorithm 1, we present pseudocode that illustrates the outline of our algorithm, GI-PPO. There are five hyperparameters in the algorithm, which are $\alpha_0, \beta, \delta_{det}, \delta_{oorr}$, and $\max(\alpha)$. We present specific values for these hyperparameters for our experiments in Appendix A.4.2.

2.5 Experimental Results

In this section, we present experimental results that show our method’s efficacy for various optimization and complex control problems. To validate our approach, we tested various baseline methods on the environments that we use. The methods are as follows:

- LR: Policy gradient method based on LR gradient.
- RP: Policy gradient method based on RP gradient. For physics and traffic problems, we adopted a truncated time window of [14] to reduce variance.
- PPO: Proximal Policy Optimization [36].
- LR+RP: Policy gradient method based on interpolation between LR and RP gradient using sample variance [50].

Algorithm 1 GI-PPO

```
1:  $\alpha \leftarrow \alpha_0$  ▷ initial value
2:  $\beta \leftarrow$  constant multiplier ( $> 1$ ) for updating  $\alpha$ 
3:  $\delta_{\text{det}}, \delta_{\text{oorr}} \leftarrow$  constant thresholds
4:  $B \leftarrow$  experience buffer
5: while training not ended do
6:   Clear  $B$ 
7:   while not collected enough experience do
8:     Collect transitions  $\{s_t, \epsilon_t, a_t, r_t, s_{t+1}\}$  and add to  $B$ 
9:   end while
10:  Estimate advantages  $A$  for all  $(s_i, a_i)$  in  $B$  using Eq. A.2
11:  Estimate advantage gradients  $\frac{\partial A}{\partial a}$  for all  $(s_i, a_i)$  in  $B$  using Eq. A.3
12:  For current  $\alpha$ , approximate the  $\alpha$ -policy by minimizing Eq. 2.10
13:  Variance
14:  Estimate  $\det(I + \alpha \cdot \nabla^2 A)$  for each  $(\epsilon_i, s_i, a_i)$  using Lemma 2.4.4
15:  Compute sample minimum  $\psi_{\min}$  and maximum  $\psi_{\max}$ 
16:  Bias
17:  Evaluate expected additional return (Eq. A.4) to obtain  $R_\alpha$ 
18:  Out-of-range ratio
19:  Evaluate out-of-range ratio (Eq. 2.12) to obtain  $R_{\text{oorr}}$ 
20:  if  $\psi_{\min} < 1 - \delta_{\text{det}}$  or  $\psi_{\max} > 1 + \delta_{\text{det}}$  or  $R_\alpha < 0$  or  $R_{\text{oorr}} > \delta_{\text{oorr}}$  then
21:     $\alpha \leftarrow \alpha / \beta$ 
22:  else
23:     $\alpha \leftarrow \alpha \cdot \beta$ 
24:  end if
25:   $\alpha \leftarrow \text{clip}(\alpha, 0, \max(\alpha))$ 
26:  Perform PPO update by maximizing the surrogate objective in Eq. A.6
27: end while
```

Table 2.1: Average maximum reward ($\uparrow$) for function optimization problems.

Problem	LR	RP	PPO	LR+RP	GI-PPO
Dejong (1)	-1.24×10^{-6}	-1.42×10^{-8}	-5.21×10^{-5}	-6.36×10^{-8}	-3.84×10^{-10}
Dejong (64)	-0.0007	-9.28×10^{-7}	-0.0011	-3.05×10^{-6}	-1.04×10^{-6}
Ackley (1)	-1.2772	-0.4821	-0.2489	-1.2255	-0.0005
Ackley (64)	-0.6378	-0.0089	-0.1376	-0.0326	-0.0036

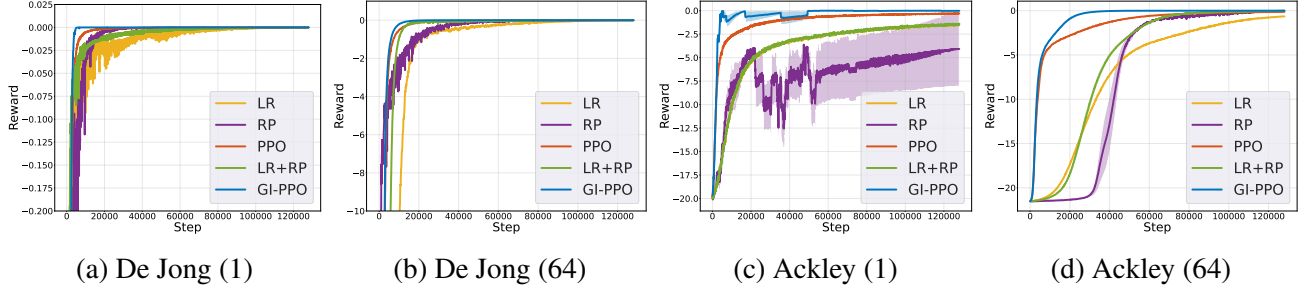


Figure 2.2: Optimization curves for Dejong’s and Ackley’s function of dimension 1 and 64.

- PE: Policy enhancement scheme of [53], for physics environments only.
- GI-PPO: Our method based on Section 2.4.3.

Please refer Appendix A.4 to learn about implementation details of these baseline methods, network architectures, and hyperparameters that we used for all of the experiments.

We have implemented our learning method using PyTorch 1.9 [90]. As for hardware, all experiments were run with an Intel® Xeon® W-2255 CPU @ 3.70GHz, one NVIDIA RTX A5000 graphics card, and 16 GB of memory. Experiment results are averaged across 5 random seeds.

2.5.1 Function Optimization Problems

We start with how different learning methods find the maximum point for a given analytical function. We ran these experiments because RL algorithms can be thought of as function optimizers through function sampling [29]. These problems are 1-step optimization problems, where agents guess

an optimal point as an action and get the negative function value as the reward.

As we used for rendering α -policies in Figure 2.1, we use De Jong’s function and Ackley’s function for comparison, as they are popular functions for testing numerical optimization algorithms [172]. Please see Appendix A.5.1 for the details about the function definitions. In this setting, the optimum occurs at the point $x = 0$ for both of the problems. However, their loss landscapes are quite different from one another—De Jong’s function has a very smooth landscape that only has one local (global) maximum, while Ackley’s function has a very rugged landscape with multiple local maxima. Therefore, De Jong’s function represents environments where RP gradients have lower variance, while Ackley’s function represents the opposite.

Table 2.1 shows that **GI-PPO finds far better optimums than the other methods for most of the functions**, except 64-dimensional Dejong’s function, where the RP method found slightly better optimum than GI-PPO. Optimization curves in Figure 2.2 also prove that our method converges to the optimum faster than the other methods. Interestingly, the RP method achieved better results than the LR and LR+RP methods for all the functions, which contradicts our assumption that it would not work well for Ackley’s function. However, we found out that even though the RP method takes some time to find the optimum than the other methods as shown in Figure 2.2, when it approaches near-optimum, it converges fast to the optimum based on analytic gradients. Since GI-PPO adaptively changes α to favor RP gradient near optimum, it could achieve better results than the other methods. In Appendix A.3.3, we provide visual renderings that trace the change of α value during optimization to help understanding, which also shows that higher α is maintained for Dejong’s function than Ackley’s function, which aligns with our initial assumption. Also note that even though LR+RP also takes advantage of the RP method, and thus achieved a better convergence rate than the two methods when it is far from optimum, it did not work well when it comes to near-optimum.

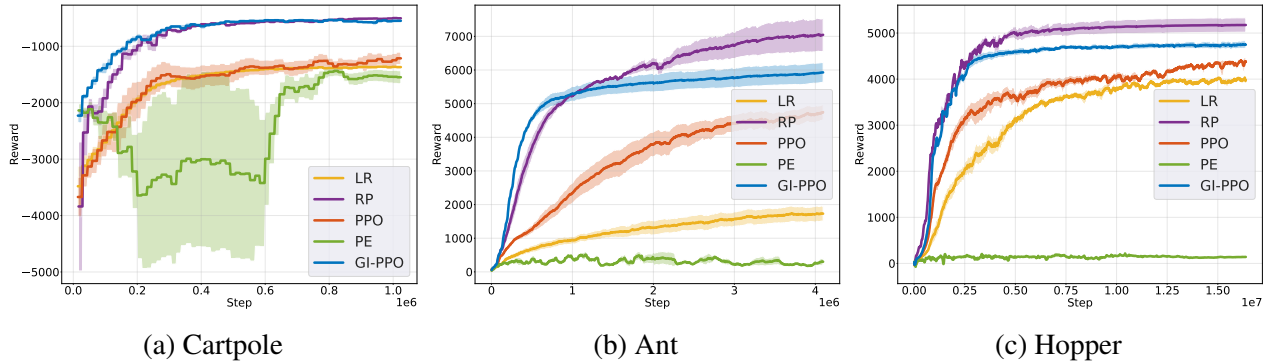


Figure 2.3: Learning graphs for 3 problems in differentiable physics simulation: Cartpole, Ant, and Hopper [14]. In these environments, RP gradients already exhibit lower variance without much bias than LR gradients, because of variance reduction scheme of [14].

2.5.2 Differentiable Physics Simulation

Next, we conducted experiments in differentiable physics simulations. We used Cartpole, Ant, and Hopper environments implemented by [14] for comparisons. However, since the current implementation does not support multiple backpropagations through the same computation graph for these environments, we could not test the LP+RP method for them. Instead, we tried to faithfully implement another policy enhancement (PE) strategy of [53] as a comparison, as it showed its efficacy in differentiable physics simulations using analytical gradients.

In Figure 2.3, we can see that GI-PPO converged to a better policy much faster than the baseline PPO for every environment, which validates our approach. However, the RP method achieved the best results for Ant and Hopper, while GI-PPO converged to the optimal policy slightly faster for Cartpole. To explain this result, we’d like to point out that α is upper bounded by out-of-range-ratio in our approach (Section 2.4.3.1). Therefore, even if the RP gradient is very effective, we cannot fully utilize it because we have to use PPO to update the policy up to some extent as a safeguard. In the Ant environment, we could observe that it was the major bottleneck in training—in fact, GI-PPO could achieve better results than RP when we increased the ratio from 0.5 to 1.0 (Appendix A.3.4). However,

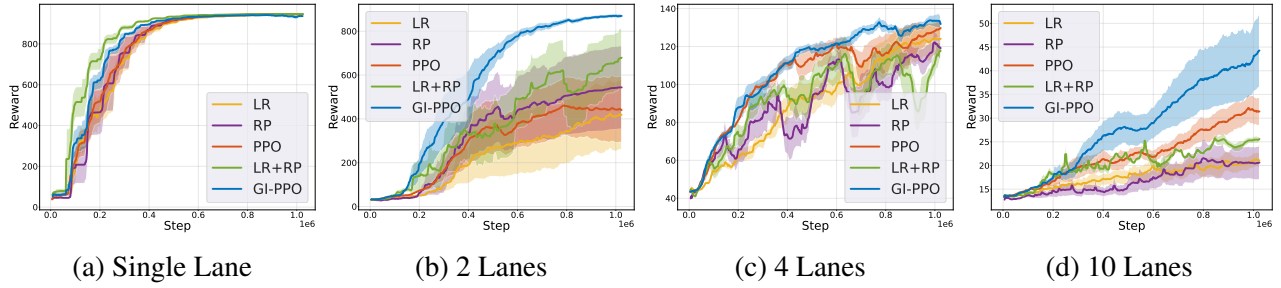


Figure 2.4: Learning graphs for traffic pace car problem. Each problem has different number of lanes and following human vehicles, which represent *discontinuities* and *dimension of the problem*, respectively.

when the out-of-range ratio equals 1.0, it becomes more likely that we cannot use PPO as a safeguard, which contradicts our design intention. When it comes to the Hopper environment, we found that the variance of proper α values over time was large, so our strategy could not conform to it properly. Overcoming these limitations would be an interesting future work to extend our work.

2.5.3 Traffic Problems

Here we demonstrate the effectiveness of GI-PPO in an environment where continuous and discrete operations coexist, which leads to highly biased analytical gradients. We suggest a mixed-autonomy traffic environment as a suitable benchmark, which has previously been explored as a benchmark for gradient-free RL algorithms [118, 173]. In this paper, we use the pace car problem, where a single autonomous pace car has to control the speed of the other vehicles via interference. The number of lanes, which represent the discontinuities in gradients, and the number of following human vehicles are different for each problem. Please see Appendix A.5.2 for the details of this environment, and why the analytical gradients are biased.

In Figure 2.4, we can observe that GI-PPO exhibits a faster convergence rate and converges to better policy overall compared to all the other methods in 2, 4, and 10-lane environments. In a single-lane environment, however, LR+RP showed a better convergence rate than GI-PPO. Note that the RP

method still performs well for 2 and 4-lane environments just as other baseline methods, even if there were multiple lanes. We assume this is because the RP method also leverages a state value function, which can provide unbiased analytical gradients in the presence of discontinuities. However, it does not perform well in the 10-lane environment, where discontinuities are excessive compared to the other environments.

This result shows that **GI-PPO can still leverage analytical gradients, even if they were highly biased for policy learning**. Even though LP+RP, which is based on a similar spirit as ours, also exhibited some improvements over LP and RP, *our method showed better results for more difficult problems because baseline PPO offers more stable learning than baseline LR*. Also, the computational cost of GI-PPO was a little more than RP, while that of LR+RP was much more expensive (Appendix A.7).

2.6 Conclusions

We introduced a novel policy learning method that adopts analytical gradients into the PPO framework. Based on the observation that the RP gradient leads our current policy to its α -policy, we suggested approximating it explicitly, which allows us to indirectly estimate variance and bias of analytical gradients. We suggested several criteria to detect such variance and bias and introduced an algorithm that manipulates α adaptively, which stands for the strength of analytical gradients in policy updates. In our experiments of diverse RL environments, we successfully showed that our method achieves much better results than the baseline PPO by adopting analytical gradients. Even though the interpolated gradient of LR and RP gradient is based on similar motivations, our method performed much better, thanks to the baseline PPO’s stable learning.

Limitations Despite GI-PPO’s promising results, there are some remaining issues. First, even if we use a large α , we may not fully approximate the corresponding α -policy in a limited number of optimization steps, which may give a false signal for our strategy. We can upper bound the maximum α as we did here, but a more rigorous optimization process may be helpful. This is also related to improving the strategy to control α adaptively, which would be a promising future work.

Also, as discussed in Section 2.5.2, our method is tightly bound to PPO—that is, even when analytical gradients are much more useful as illustrated in differentiable physics problems, we cannot fully utilize them if they exit the PPO’s bound. Even though it is more suitable for stable updates, it could result in a worse convergence rate. If we could adjust PPO’s clipping range dynamically, or detect environments where RP gradients are much more reliable, we would be able to overcome this issue.

Finally, computational costs can be further reduced. To utilize analytical gradients, we need backpropagation, which usually requires more time than forward steps. This is the reason why the learning methods based on analytical gradients require longer training time than the others (Appendix A.7). When the time window gets longer, the cost grows accordingly. However, as we have shown in our traffic experiments, our method works even when gradients are biased. Therefore, it would be an interesting research direction to see if using a very short time window for backpropagation, which produces more biased gradients, would be worthy of exploring for improved computational efficiency.

Taken together, these limitations highlight that effectively leveraging analytical gradients in reinforcement learning depends not only on how gradients are incorporated into policy updates, but also on how accurately and consistently system dynamics are represented over time. In particular, the reliance on finite backpropagation windows and policy-level constraints motivates a closer examination of how

temporal structure and dynamics modeling influence the quality and utility of gradient information. In the next chapter, we therefore shift focus from policy-level gradient integration to learning time-aware world models that explicitly capture system dynamics across multiple time scales.

Chapter 3: Time-Aware World Model for Adaptive Prediction and Control

As discussed in the previous chapter, the effectiveness of gradient-informed learning critically depends on how accurately system dynamics and their temporal sensitivities are represented. This chapter addresses this challenge by introducing the Time-Aware World Model (TAWM), a model-based formulation that explicitly conditions on the time-step size, Δt , to capture system dynamics across multiple temporal resolutions. By learning from a diverse range of time scales rather than a fixed sampling rate, this approach provides a principled foundation for modeling dynamics and their associated sensitivities in complex control tasks.

3.1 Introduction

Deep reinforcement learning (DRL) has recently demonstrated human-level or even expert-level capabilities on many highly complex and challenging problems, such as Go [174, 175], Chess and Shogi games [176], and StarCraft II video game [177]. DRL is also effectively adopted in a broad range of applications, including robotics [72, 178, 179], autonomous vehicles [180, 181], and challenging control tasks where classical approaches fail to deliver satisfactory performance [182–184]. Beyond model-free reinforcement learning (RL) methods, where an agent directly maps observation o_t to action a_t [29, 35, 36, 185], there has been growing interest in model-based RL (MBRL). Unlike model-free approaches, MBRL constructs a model $\mathcal{M}$ that captures the underlying task dynamics [50, 65, 186–188], enabling the agent to plan actions using the learned *world model* $\mathcal{M}$. These model-based methods have gained traction over their model-free counterparts due to their superior sample efficiency and enhanced generalization capabilities [66–71].

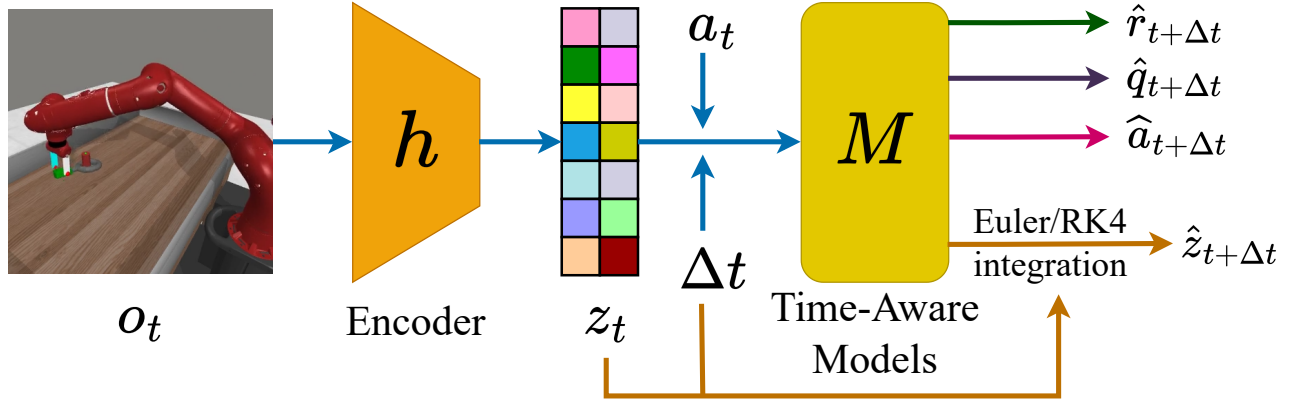


Figure 3.1: **Overall framework of our time-aware world model.** An encoder h encodes the given observation o_t into a latent vector z_t , which is then fed into various models with action a_t and *time step size* Δt to estimate values for action planning.

Despite the remarkable success of world models in various control tasks, a critical factor in handling dynamical systems – the time step size, Δt – has been largely overlooked in existing research. Specifically, the dynamics model $\mathcal{D} : (s_t, a_t) \rightarrow s_{t+1}$, a fundamental component of the world model, governs state transitions from the current state s_t to the next state s_{t+1} . Conventional approaches train $\mathcal{D}$ using experience tuples (o_t, a_t, o_{t+1}, r_t) collected from interactions with the environment at a *fixed* time step size. However, this existing practice presents three key limitations [189]:

1. **Temporal resolution overfitting:** In current training pipelines, the observation time step Δt are often fixed (e.g., $\Delta t = 2.5ms$ or frequency $f = 400Hz$). While smaller Δt values stabilize simulations and prevent system aliasing, world models trained exclusively at a single, fixed Δt often suffer significant performance degradation when deployed in real-world scenarios with different observation rates (e.g. $\Delta t = 20ms$ or $f = 50Hz$) due to compounding errors [190,191] and computational costs of roll-out predictions. This discrepancy presents a major challenge in extending the applicability of world models beyond simulated environments.
2. **Inaccurate system dynamics:** Training the dynamics model $\mathcal{M}$ on a fixed time step Δt can lead to overfitting, as the model may not capture the true underlying dynamics of the task without

conditioning on Δt . This can result in inaccurate state transitions and diminished generalization capabilities.

3. **Inefficient dynamics learning:** Although small Δt values are essential for numerical stability and for capturing high-frequency dynamics, relying exclusively on fine-grained time steps results in inefficient sampling. Real-world systems often exhibit multiscale behavior, with some components evolving rapidly while others change more slowly. Sampling solely at a high frequency captures redundant information about the slower components, wasting computational resources and training data. This issue is exacerbated in systems with widely varying time scales, where no single sampling rate is optimal for every dynamical component.

Recent work [192, 193] has explored temporal aspects of world models, but primarily to improve long-horizon predictions through roll-outs of fixed- Δt models. Our approach instead conditions the world model directly on Δt , enabling single-step predictions across arbitrary temporal intervals (Figure 3.1). Although previous methods can, in principle, accommodate variable time steps through repeated application, that strategy compounds errors and is computationally inefficient. By sampling strategically across multiple time scales, our method lets the model learn dynamics at different frequencies simultaneously, without increasing the number of training samples. Consequently, it generalizes effectively across varying observation rates – crucial for simulation-to-reality transfer – while maintaining superior sample efficiency compared with fixed- Δt approaches.

In this work, We focus on the following question: *How can we efficiently train a world model $\mathcal{M}$ to capture the underlying task dynamics across varying time step sizes Δt without increasing sample complexity?*

To answer this, we introduce a time-aware world model, $\mathcal{M}^{\text{TA}}$. Unlike earlier world models $\mathcal{M}$,

our model conditions both the dynamics model and the reward function on Δt (Figure 3.1), because these depend on the temporal gap between consecutive states. We construct $\mathcal{M}^{\text{TA}}$ by adapting the TD-MPC2 world model [71] and incorporating the fourth-order Runge–Kutta (RK4) method [194] to stabilize learning at large Δt values (Section 3.4.1). We likewise modify the value model to accept Δt as an additional input. Both models are trained on Δt values log-uniformly sampled from a predefined interval.

Although one might expect $\mathcal{M}^{\text{TA}}$ to require more training samples than $\mathcal{M}$ because of the additional parameter Δt , this is not necessarily the case. According to the Nyquist–Shannon sampling theorem [195, 196], a signal with highest frequency f can be perfectly reconstructed by sampling at any frequency just above $2f$. Thus, if the observation rate greatly exceeds $2f$, the excess data are redundant – they contribute little to training the world model. A physical environment typically comprises multiple dynamical subsystems operating at different characteristic frequencies (Section 3.3.2.1). By training with a mixture of time-step sizes, we expose the model to a range of effective sampling frequencies, enabling each subsystem to be learned more efficiently (Section 3.3.2.3).

Inspired by the Nyquist–Shannon sampling theorem (Section 3.3.2.2), *we prove that our time-aware model trained on observation data sampled at a mixture of time steps achieves substantially better performance in learning world dynamics across different time steps at inference time, while using the same training budget as the baseline.* We demonstrate these results on a variety of control problems in Meta-World [197] and PDE-control environments [198]. Our contributions are summarized as follows:

1. We underscore the importance of a time-aware world model (TAWM) that conditions its dynamics modeling on the time step size Δt , a fundamental quantity in any dynamical system.

By explicitly incorporating Δt , the model learns to capture the underlying task dynamics across

a broad spectrum of step sizes. Our model’s ability to generate a *one-step prediction* of the next state conditioned on Δt *mitigates the well-known problem of compounding errors*. This approach is particularly suitable for real-world learning and control, where the observation rate may vary – or be substantially lower than – the default observation rate used for training.

2. Motivated by the Nyquist–Shannon sampling theorem and by the fact that real-world dynamics comprise many subsystems operating at different – and often unknown – frequencies, we propose a mixture-of-time-step training framework for TAWM that performs well across a range of observation rates at inference time, *without increasing the number of training steps*. This perspective – training a world model with varying sampling rates – offers a new avenue for developing more efficient training strategies.
3. Empirically, we demonstrate that our time-aware world model can solve a range of control tasks at various observation rates *without increasing either the amount of data or the number of training steps*. This capability helps narrow the gap between simulation environments and real-world control problems.

3.2 Related Work

Although model-free RL algorithms have gained popularity for their impressive performance, their inherent sample inefficiency limits their applicability to many real-world problems [29,33,35,36,168]. Several works mitigate this limitation by integrating low-variance analytical gradients into the policy-learning process [14,52,199].

In contrast, model-based RL (MBRL) tackles sample inefficiency at its root by training a dynamics model that *simulates* the environment, enabling agents to *predict* future states and action out-

comes [64]. Because prediction is unconstrained by real-time sampling, MBRL is inherently more sample efficient. MBRL methods vary mainly in (1) how they define the world model $\mathcal{M}$ and (2) how they exploit $\mathcal{M}$ for training or planning. Historically, Gaussian processes (GPs) [50, 65] were widely used, but modern work favors neural networks [66–71] for their greater representational power. Among these, Dreamer and its variants [67–69, 72] train policies directly inside the learned model, whereas model-predictive-control (MPC) methods [70, 71] rely on action planning.

However, these approaches paid little attention to the temporal component when designing the world model $\mathcal{M}$, despite its importance in dynamical systems. To address its limitation in long-horizon prediction, Multi-Time-Scale World Models (MTS3) [192] explicitly leverages temporal gaps when learning task dynamics. MTS3 captures state transitions over different prediction horizons H but uses a single fixed Δt during training; it therefore represents only two discrete time scales – *fast* and *slow* dynamics corresponding to Δt and $H\Delta t$. In contrast, we incorporate a continuous-valued Δt directly into the model, enabling single-step predictions across a range of inference step sizes. This allows our model to handle large temporal gaps (e.g., $\Delta t = 30ms$) that would otherwise require 12–20 smaller steps in a fixed-step model. Our approach is *architecture agnostic*: it builds on the TD-MPC2 framework [71] yet crucially differs by explicitly conditioning the world model on Δt and training on a mixture of step sizes, rather than assuming a fixed step throughout.

3.3 Background and Motivations

3.3.1 Model-Based Reinforcement Learning

We formulate the control problem as a Markov decision process (MDP) $\langle S, A, P, r, \gamma \rangle$, where S is the state space, A is the action space, $P : S \times A \times S \rightarrow \mathbb{R}$ is the (stochastic) state-transition function,

$r : S \times A \rightarrow \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. We refer to P and r as the *ground-truth* models, because in Section 3.4.1 we learn approximations of them within our world model.

The goal of reinforcement learning is to obtain a policy or planner π that maximizes the expected discounted return along a trajectory $\tau = \{s_0, a_0, \dots, s_{H-1}, a_{H-1}, s_H\}$ of length H :

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \dots \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \quad (3.1)$$

In model-based RL (MBRL), we train a world dynamics model consisting of a state-transition function $d_\phi : S \times A \rightarrow S$ and a reward function $r_\phi : S \times A \rightarrow \mathbb{R}$. The learned world model can then be exploited in multiple ways to derive a policy, for instance by using a planner such as TD-MPC2. Section 3.4.1.2 details our model formulation and the training pipeline for the time-aware world model.

3.3.2 Theoretical Motivations

Here we provide theoretical motivations to explain the sample efficiency of using a mixture of time step sizes during the training process of the time-aware world model.

3.3.2.1 Multi-scale Dynamical Systems

In many control problems, the environmental dynamics can be decomposed into multiple subsystems, each evolving on a different temporal scale [200]. In other words, these subsystems can be described by distinct functions, each with its own highest frequency. Formally, consider a general dynamical system $\dot{x} = f(x, u, t)$, where x , u , and t denote the state, control input, and time, respectively.

Applying the Euler integration method, the next state x' is

$$x' = x + f(x, u, t) \cdot \Delta t \quad (3.2)$$

where Δt is the time-step size. From a multi-scale perspective, this update can be rewritten as

$$x' = x + \sum_i f_i(x, u, t) \cdot \Delta t \quad (3.3)$$

where $f_i(x, u, t)$ represents the dynamics of subsystem i . Each subsystem may evolve at its own temporal scale and therefore have its own highest frequency $f_{\max}^i$.

3.3.2.2 Nyquist-Shannon Sampling Theorem

The Nyquist–Shannon sampling theorem states that a signal must be sampled at a rate of at least twice its highest frequency to avoid information loss (i.e., to prevent aliasing): $f_{\text{sample}} > 2f_{\max}$ [195]. Here, f_{sample} is the observation rate, and $f_{\max}$ is the highest frequency of the environment’s dynamics in the MBRL context. If the observation rate is too low – such that $f_{\text{sample}} < 2f_{\max}$ – we lose important dynamic details due to the large temporal gap Δt . This loss causes high-frequency components to fold back into lower frequencies, leading to inaccurate dynamics learned by the world model [201]. Although a higher f_{sample} enables more accurate reconstruction of the environment’s dynamics, an excessively high rate leads to oversampling, introducing redundant data that increase sample complexity and reduce learning efficiency. Therefore, choosing an appropriate observation rate f_{sample} is crucial for balancing modeling accuracy and sample efficiency.

3.3.2.3 Simultaneously training on multiple temporal resolutions

To better align observation sampling with task dynamics across subsystems operating at different frequencies, we propose simultaneously training the world model at multiple temporal resolutions by varying the observation rate f_{sample} (i.e., varying Δt) during training. As shown in Equation 3.3, the overall dynamics comprise several subsystems $f_i(\cdot)$, each with its own maximum frequency f_i^{max} . According to the Nyquist–Shannon sampling theorem, each subsystem is learned most efficiently at a distinct f_{sample} . By randomly varying f_{sample} during training, we avoid undersampling high-frequency components and oversampling low-frequency ones, thereby training the subsystems $f_i(\cdot)$ more efficiently. Consequently, our time-aware world model learns the underlying task dynamics at multiple temporal resolutions without needing additional data (Section 7.5).

3.4 Methodology

In this section, we present our time-aware model formulation and training pipeline, designed to learn a world model that performs well across a range of observation rates. We introduce a novel time-aware training method that can be seamlessly integrated into any existing world-model architecture, enhancing robustness to observation-rate variations at inference time. We adopt TD-MPC2 [71] as our baseline and adapt its architecture to train time-aware world models for various control tasks. We begin with a high-level overview of TD-MPC2 and its key architectural components. The overall framework is depicted in Figure 3.1.

3.4.1 Model Architecture

3.4.1.1 Baseline World Model

TD-MPC2 [71] is a model-based RL algorithm that captures task dynamics in a latent space – an “implicit” world model. Unlike reconstruction-based architectures [66, 69], TD-MPC2 omits a decoder that maps latent representations back to raw observations. Recovering latent states in the high-dimensional observation space is computationally expensive and often unnecessary, as many elements are irrelevant to control. Once the latent dynamics are learned, TD-MPC2 rolls out latent predictions for local trajectory optimization with planning algorithms such as MPC [70, 71]. The baseline TD-MPC2 model comprises five key components:

$$\begin{array}{ll} \text{Encoder:} & z_t = h(o_t) \\ \text{Latent dynamic:} & \hat{z}_{t+1} = D(z_t, a_t) \\ \text{Reward:} & \hat{r}_t = R(z_t, a_t) \\ \text{Terminal value:} & \hat{q}_t = Q(z_t, a_t) \\ \text{Policy prior:} & \hat{a}_t = p(z_t) \end{array}$$

where o_t is the raw observation at time step t , z_t is its latent encoding, $\hat{z}_{t+1}$ is the predicted next latent state, $\hat{r}_t$ is the predicted immediate reward, $\hat{q}_t$ is the estimated Q -value of the current state–action pair, and $\hat{a}_t$ is the action sampled from the policy for the current state. At inference time, the Model Predictive Path Integral (MPPI) planner – an instance of Model Predictive Control (MPC) – is used for planning and action generation. All five component models are implemented using multilayer perceptrons (MLPs).

To train TD-MPC2, we maintain a replay buffer $\mathcal{B}$ that stores trajectories $(o_t, a_t, o_{t+1}, r_t)_{t=0}^H$ collected from the environment after each episode of length H . At the end of every episode, the model parameters are updated with batches randomly sampled from $\mathcal{B}$. The encoder h , dynamics model D ,

reward model R , and terminal-value model Q are optimized jointly using a self-supervised consistency loss, a supervised reward loss, and a supervised temporal-difference loss for the terminal value. See [71] for details. The agent then resumes interaction with the environment, gathering additional data to augment $\mathcal{B}$ for subsequent training.

3.4.1.2 Time-Aware World Model

One notable limitation of TD-MPC2 and other state-of-the-art world models is that they ignore the effect of the observation rate Δt at inference time. To introduce time awareness, we condition every component of the world model on Δt , as described below:

$$\begin{array}{ll}
\text{Encoder:} & z_t = h(o_t) \\
\text{Latent dynamics:} & \hat{z}_{t+\Delta t} = z_t + d(z_t, a_t, \Delta t) \tau(\Delta t) \\
& \text{where } \tau(x) = \max(0, \log_{10} x + 5) \\
\text{Reward model:} & \hat{r}_t = R(z_t, a_t, \Delta t) \\
\text{Value model:} & \hat{q}_t = Q(z_t, a_t, \Delta t) \\
\text{Policy prior:} & \hat{a}_t = p(z_t, \Delta t)
\end{array}$$

Our time-aware model formulation is *architecture-agnostic* and can be readily integrated into any state-of-the-art world model. Because the observation encoder merely maps raw observations to the latent space, without modeling dynamics, it is *not* conditioned on Δt . All other components that depend on the system’s dynamics *are* conditioned on the time step size by taking Δt as an explicit input.

While the baseline latent dynamics model D maps a state–action pair (z_t, a_t) directly to the next latent state z_{t+1} , we reformulate D using Euler integration:

$$\hat{z}_{t+\Delta t} = D(z_t, a_t, \Delta t) = z_t + d(z_t, a_t, \Delta t) \cdot \tau(\Delta t),$$

where $d(\cdot)$ is implemented as an MLP. This formulation enforces the intrinsic property

$$z_{t+\Delta t}|_{\Delta t=0} = z_t, \forall z_t, a_t.$$

Thus, instead of learning the transition function D directly, TAWM learns the latent-state derivative (gradient) function d , which is itself conditioned on Δt to capture higher-order effects, and then advances the state by a single Euler step.

State Transition Rather than integrating over the raw time step Δt , our dynamics model integrates using

$$\tau(\Delta t) = \max(0, \log_{10}(\Delta t) + 5).$$

The possible values of Δt span several orders of magnitude – from a minimum of about 10^{-3} s to a maximum of roughly 5×10^{-2} s – creating numerical challenges. Because latent vectors change only slightly between steps, the dynamics model $d(z_t, a_t, \Delta t)$ must scale appropriately across this wide range. Empirically, we observed convergence failures in some tasks (e.g., `mw-assembly`) when employing linear Δt integration. By allowing the latent state to evolve with respect to the logarithm of the time step via $\tau(\Delta t)$, we effectively normalize Δt to a narrower range, mitigating numerical issues and enabling more stable learning.

Before proceeding, we note that we experimented with two popular integration methods for our dynamics model: **(1)** Euler and **(2)** fourth-order Runge–Kutta (RK4), even though only Euler integration was introduced earlier. See Appendix B.1 for RK4 implementation details. We include RK4 because it is broadly applicable to both simple linear and complex nonlinear systems and is standard in physical simulation. The integration method thus serves as a tunable hyperparameter for

Algorithm 2 Time-Aware World Model Training Paradigm

```
1: Initialize task environment  $\mathcal{E}$ 
2: Initialize time-aware world model  $\mathcal{M}^{TA}$ 
3: Initialize experience buffer  $\mathcal{B} \leftarrow \emptyset$ 
4: repeat
5:   for each episode do
6:     Sample  $\Delta t \sim \text{Log-Uniform}(\Delta t_{\min}, \Delta t_{\max})$ 
6:      $\triangleright$  Meta-World:  $\Delta t_{\min} = 0.001, \Delta t_{\max} = 0.05$  ( $\Delta t_{\text{default}} = 0.0025$ )
6:      $\triangleright$  Log-Uniform or Uniform sampling can be used
7:      $step \leftarrow 0$ 
8:     while  $step < H$  do
9:        $a_t \leftarrow \mathcal{M}^{TA}.\text{act}(o_t, \Delta t)$ 
10:      Execute  $a_t$  in  $\mathcal{E}$  and observe  $(o_{t+\Delta t}, r_t)$ 
11:      Add  $(o_t, a_t, o_{t+\Delta t}, r_t, \Delta t)$  to  $\mathcal{B}$ 
12:      Sample minibatch  $\{(o_t, a_t, r_t, o_{t+\Delta t}, \Delta t)\}_{1:B} \sim \mathcal{B}$ 
13:      Update  $\mathcal{M}^{TA}$  using the sampled minibatch
14:       $step \leftarrow step + 1$ 
15:     end while
16:   end for
17: until total training steps reach  $N$ 
18: return  $\mathcal{M}^{TA}$ 
```

maximizing TAWM’s performance and learning efficiency across control tasks.

For most Meta-World robot-manipulation tasks, the dynamics are simple enough to be well approximated by Euler integration. Our ablation study shows that TAWM with Euler integration outperforms its RK4 counterpart on the majority of Meta-World tasks, indicating that Euler suffices in these settings (Figure 3.3). By contrast, RK4’s benefits become more pronounced for tasks with complex dynamics, such as those in PDE-control environments (Figure 3.3).

3.4.2 Training Procedure Using a Mixture of Δt

Algorithm 2 summarizes our training procedure, in which we vary the observation rate to encourage the model to learn the underlying dynamics at multiple temporal resolutions. The world model acquires a spatiotemporal representation of the environment by sampling observations from the var-

ious dynamical subsystems at specific points in time and space. According to the Nyquist–Shannon sampling theorem, a signal must be sampled at a rate of at least $1/(2f_{\max})$, where $f_{\max}$ is the highest frequency present in the band-limited signal. Because there is no systematic way to determine $f_{\max}$ for every subsystem, we instead sample observations at several temporal rates to capture the underlying dynamics more effectively.

At the start of each episode (Algorithm 2), we sample Δt from a log-uniform distribution and set the observation/control rate to $1/\Delta t$. A log-uniform distribution assigns equal probability mass to every octave of time scales, yielding good coverage across multiple orders of magnitude. This is especially useful when $\Delta t_{\max} \geq \frac{1}{2f_{\max}}$, where $f_{\max}$ is the highest frequency of the task dynamics. For tasks with dynamics slow enough to be captured by $\Delta t_{\max}$, however, uniform sampling may outperform log-uniform sampling, particularly at larger Δt values (Figure 3.7).

3.4.3 Theoretical Analysis of TAWM’s Sample Efficiency

We present theoretical insights into TAWM’s sample efficiency, showing that it can be trained to capture task dynamics across temporal scales *without increasing the number of samples or training steps and without sacrificing performance*. We begin with two core assumptions:

1. **Model capability:** Because our time-aware training framework is model-agnostic, it inherits the limitations of the underlying world-model architecture. We assume that this architecture has enough representation power to capture the system’s complex dynamics.
2. **Sampling frequency:** We focus our analysis on those subsystems and time intervals for which $\Delta t_{\max}$ can effectively capture the underlying dynamics. We assume that the empirically determined range $[\Delta t_{\min} = 0.001s, \Delta t_{\max} = 1s]$ adequately captures the behavior of most subsys-

tems.

In the following discussion, we denote by $\bar{f}$ the ground-truth dynamics function, which satisfies the equation:

$$z_{t+\Delta t} = z_t + \bar{f}(z_t, a_t, \Delta t) \cdot \Delta t.$$

Also, we say that an environment's dynamics can be *fully captured* with a time step $\Delta \bar{t}$ if its dynamics function $\bar{f}$ satisfies:

$$\|\bar{f}(z_t, a_t, \Delta \bar{t}) - \bar{f}(z_t, a_t, \Delta t)\| < \epsilon, \forall \Delta t < \Delta \bar{t},$$

where $\epsilon \ll 1$ is a sufficiently small constant.

Then, the following lemma shows that the optimal time-aware dynamics function d^* (Section 3.4.1) can approximate dynamics for all $\Delta t < \Delta \bar{t}$ through simple interpolation.

Lemma 3.4.1 *When the environment's dynamics can be fully captured with $\Delta \bar{t}$, the optimal dynamics function d^* satisfies the following relation for any smaller time step $\Delta t < \Delta \bar{t}$:*

$$\|d^*(z_t, a_t, \Delta t) - d^*(z_t, a_t, \Delta \bar{t}) \cdot \frac{\Delta t}{\tau(\Delta t)} \cdot \frac{\tau(\Delta \bar{t})}{\Delta \bar{t}}\| < \epsilon.$$

Proof. Please see Appendix B.7.6.1.

Note that the interpolation factor $\frac{\Delta t}{\tau(\Delta t)} \cdot \frac{\tau(\Delta \bar{t})}{\Delta \bar{t}}$ is common to every training sample; regardless of z_t or a_t , this relationship holds. Based on this observation, we introduce our third core assumption as follows.

3. **Interpolation Learning:** During training, the dynamics function d effectively learns the relationship described in Lemma 3.4.1 as the interpolation factor is shared across all samples.

With this assumption, we can prove the following lemma, which states that error reductions at larger temporal scales contribute to reductions at smaller scales during training.

Lemma 3.4.2 *When the environment’s dynamics can be fully captured with $\Delta\bar{t}$, reducing the modeling error at $\Delta\bar{t}$ lowers the upper bound of the error for every $\Delta t < \Delta\bar{t}$.*

Proof: Please see Appendix B.7.6.2.

Building on this lemma, we conjecture that improvements gained at one temporal scale transfer to all smaller scales, thereby enhancing TAWM’s sample efficiency. In the next section, we empirically confirm this conjecture by demonstrating TAWM’s high sample efficiency.

3.5 Experiments

We address three key questions in our experiments: **(1)** Under the same planner, does TAWM match the baseline’s performance at the default observation rate while avoiding degradation at lower rates? **(2)** At which observation rates does TAWM outperform the baseline? **(3)** Does TAWM require more training data than the baseline? We primarily investigate these questions in Section 3.5.1 and present ablation studies on sampling strategies in Section 3.5.2.

3.5.1 Evaluations on Control Problems

To answer our main questions and evaluate the performance and learning efficiency of our time-aware world model, we conducted experiments on several control tasks from the Meta-World simulation suite, which uses a default time step of $\Delta t = 2.5ms$ (0.0025s) [197]. We tested *nine* diverse tasks,

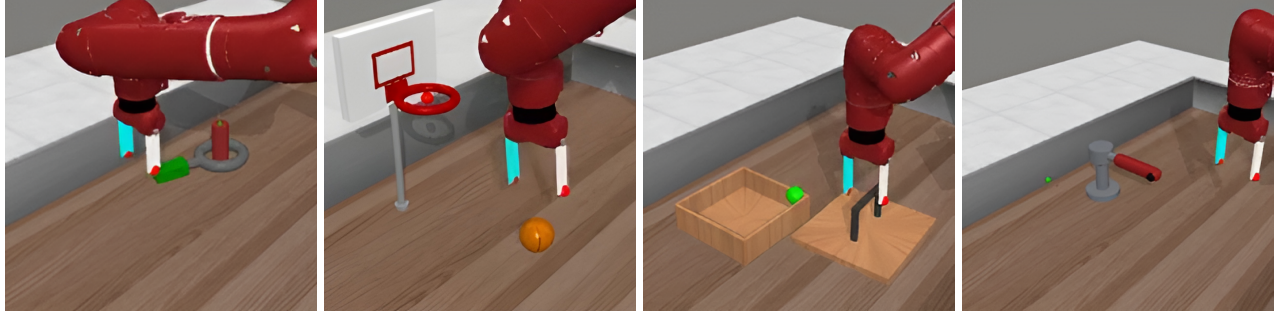


Figure 3.2: **Visualizations of the Meta-World control tasks.** From left to right: *Assembly*, *Basketball*, *Box Close*, *Faucet Open*. The default time step size of all environments is $\Delta t = 2.5ms$.

each with distinct goals and motion characteristics: Assembly, Basketball, Box Close, Faucet Open, Hammer, Handle Pull, Lever Pull, Pick Out of Hole, and Sweep Into. Representative environments are shown in Figure 3.2, and Appendix B.2 provides renderings of all nine tasks. Following [71], we report the success rate (%) as the primary metric for comparing the time-aware and baseline models on these Meta-World control tasks.

On top of that, our experiments also include three non-linear, one-dimensional PDE-control tasks from `control-gym` [198]: Burgers, Allen–Cahn, and Wave. For these tasks, we use the (continuous) total episode reward as the primary metric. Because of space constraints, we defer detailed problem descriptions, visualizations, and additional analysis of these environments to Appendix B.7.

Training Setup. We built our time-aware model on the base TD-MPC2 architecture, using the same default training hyperparameters (e.g., model size, learning rate, and horizon). As shown in Algorithm 2, we randomly varied Δt during training to create a mixture of training observation rates. Because there is no systematic way to identify the highest frequency in each task’s dynamics, we set Δt to the range $[0.001, 0.05]$ for Meta-World tasks and $[0.01, 1.0]$ for PDE-control tasks. For Meta-World tasks, each TAWM was trained for 1.5 million steps, which required roughly 40–45 hours on a single NVIDIA RTX 4000 GPU (16 GB VRAM) and 32 CPU cores. For PDE-Allen-Cahn and PDE-Wave,

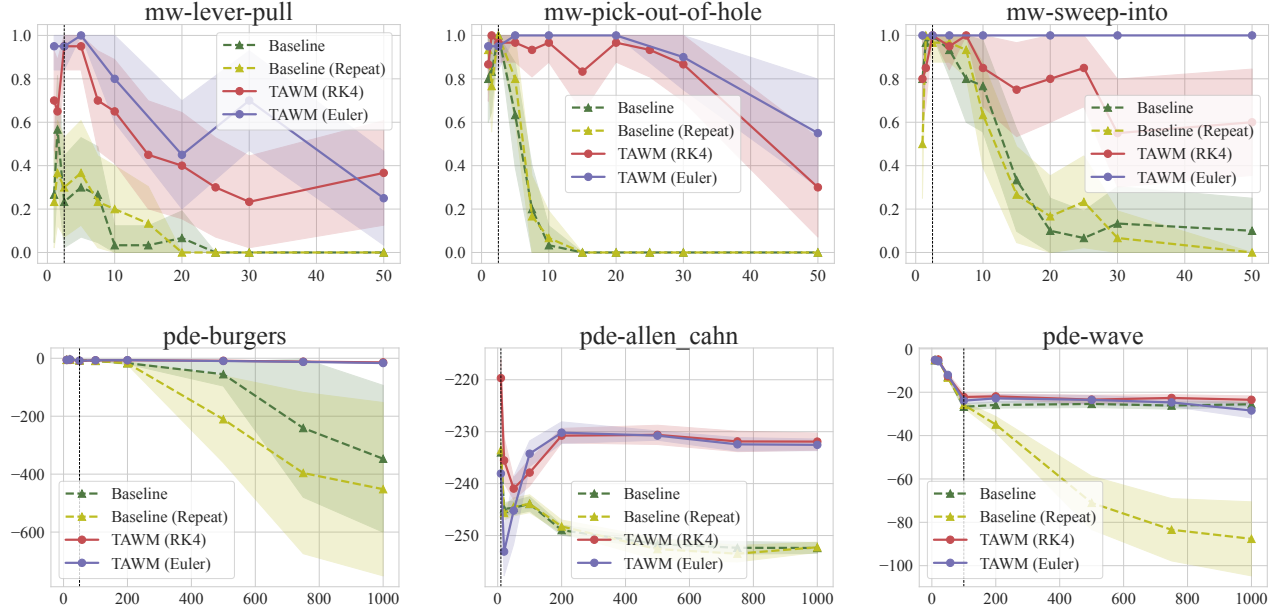


Figure 3.3: **Performance comparisons for 3 tasks from Meta-World (Up), and 3 tasks from PDE-Control (Down).** The x-axis corresponds to the evaluation Δt (in ms), and the y-axis corresponds to success rate (Meta-world) and episode reward (PDE-Control). We trained our TAWM model using either **RK4** or **Euler** integration method with log-uniform sampling strategy. The **baseline** method was trained using only default time step ($\Delta t = 2.5 ms$ for Meta-World tasks; $\Delta t = 50 ms/10 ms/100 ms$ for PDE-Burgers, PDE-Allen-Cahn, PDE-Wave, respectively), which is signified with the black dashed line in each environment. For fair comparisons, we extended the baseline method by **repeating** the same actions for the larger evaluation time steps than the default one. For instance, when the evaluation $\Delta t = 5ms$, we repeat the same action for $5/2.5 = 2$ times. Plots show mean and 95% confidence intervals over 3 seeds, with 10 evaluation episodes per seed.

the TAWM and baseline models were trained for 1M steps. For PDE-Burgers, all models were trained for 750k steps.

Performance Comparisons Across Different Δt . To assess the time-aware model’s performance at different inference-time observation rates, we evaluated it on multiple tasks with varying Δt . As shown in Figure 3.3 and Figure B.4, both TAWM variants – one using RK4 integration and the other using Euler integration – outperform the baseline (trained at a fixed $\Delta t = 2.5 ms$) across all tasks under identical hyperparameters. We observed similar gains in every other Meta-World task (Appendix B.5),

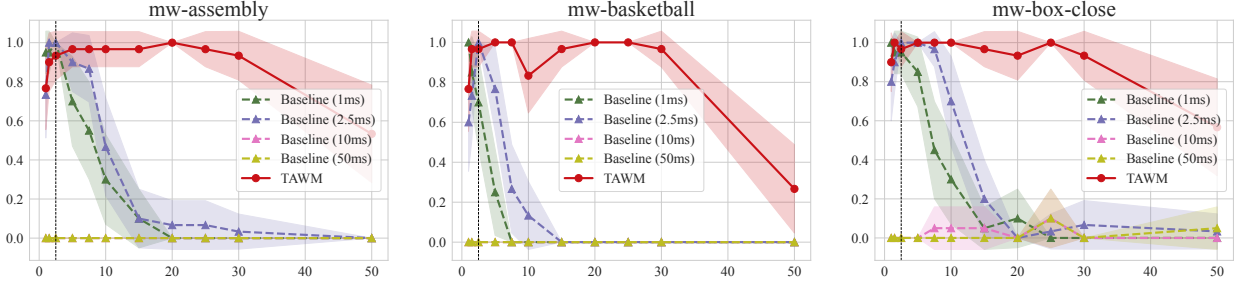


Figure 3.4: **Performance comparisons to baseline methods trained on different Δt values for Meta-World tasks.** The x-axis corresponds to the evaluation Δt , and the y-axis corresponds to the success rate. The non-time-aware baseline models were trained with different fixed Δt values (shown in the legend). Our TAWM model employs RK4 integration and log-uniform Δt sampling. **TAWM outperforms all baseline models trained with fixed time-step sizes.** When the baselines are trained only at low observation rates ($\Delta t \geq 10\text{ms}$), they fail on *all* tasks. The dashed black line indicates the default time-step size ($\Delta t = 2.5\text{ms}$). Means and 95% confidence intervals are computed over 3 seeds, each evaluated for 10 episodes.

indicating that the time-aware model effectively learns both fast and slow dynamics without increasing sample complexity. Empirically, the dynamics of some Meta-World tasks are sufficiently simple to be well approximated by TAWM with Euler integration, leading to better performance than with RK4 integration.

Effects of using Mixtures of Δt . To demonstrate the effectiveness of training the world model at multiple temporal resolutions Δt , we compare our approach with baselines trained only at fixed Δt values different from the default $\Delta t = 2.5\text{ms}$. Figure 3.4 shows that our time-aware model outperforms all baselines across three Meta-World tasks. Most notably, when baselines are trained solely at low observation rates (e.g., $\Delta t \geq 10\text{ms}$), they fail to converge and achieve zero success on every task at every evaluation rate. In contrast, by training with a mixture of time steps, our time-aware model consistently surpasses any baseline trained at a single step size, regardless of that fixed Δt . These results suggest that environmental dynamics comprise multiple subsystems, each describable as

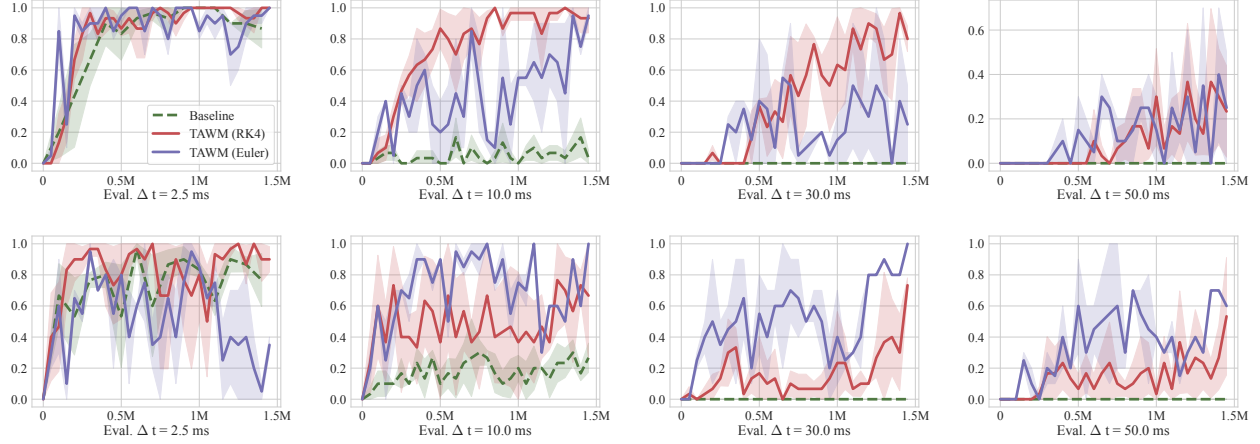


Figure 3.5: **Learning curve comparison on Meta-World Basketball (Up) and Meta-World Lever-Pull (Down).** The x-axis corresponds to the evaluation Δt (in ms), and the y-axis corresponds to the success rate. We trained our TAWM using either **RK4** or **Euler** integration method with log-uniform sampling strategy. The **baseline** method was trained using only the default time step ($\Delta t = 2.5ms$). Plots show mean and 95% confidence intervals over 3 seeds, with 10 evaluation episodes per seed.

a time-dependent, spatially parameterized function with its own highest frequency. Varying the observation rate (i.e., varying Δt), therefore enables the world model to learn these underlying subsystems more effectively.

Convergence Rate on Various Δt . To evaluate the sample efficiency of TAWM, we compare the episode success rate curves across different control tasks between our TAWM models and non-time-aware baseline models trained with a fixed default time step of $\Delta t = 2.5$ ms. We assess the success rate curves of TAWM and baseline models under different evaluation Δt values across various control tasks throughout training. In Figure 3.5, we present the learning curves for the Meta-World Basketball and Lever-Pull tasks. Despite having to learn state transitions under varying time step sizes, our TAWM with RK4 integration method converges at least as quickly as the baseline when evaluated at the default $\Delta t = 2.5$ ms – the exact time step on which the baseline was specifically trained. Compared to the non-time-aware baseline specialized at only $\Delta t = 2.5$ ms, TAWM (RK4) still converges to a higher success rate at $\Delta t = 2.5$ ms after 1.5M training steps. Both TAWM variants (RK4 and Euler) reach

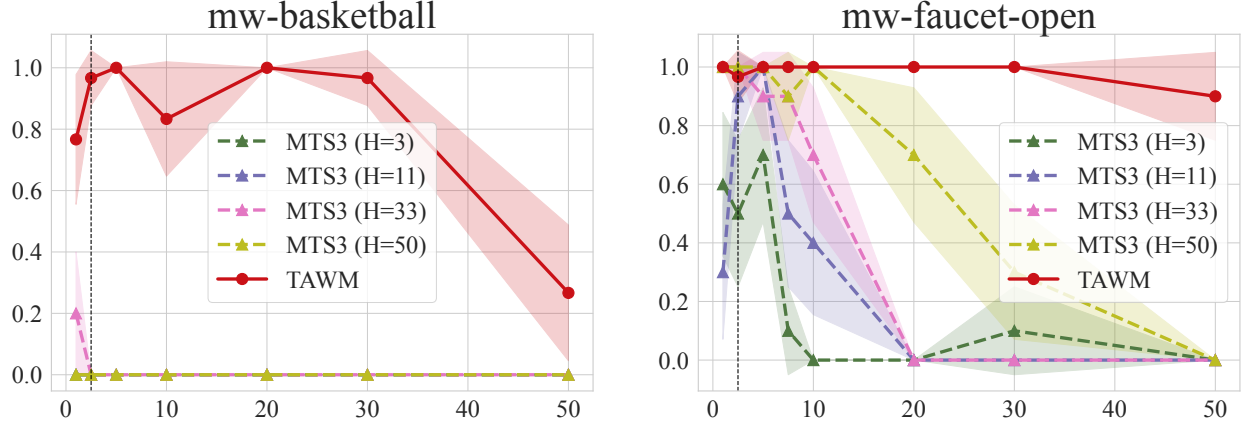


Figure 3.6: **Performance comparisons to MTS3 across different evaluation Δt 's on Meta-World tasks.** The x-axis corresponds to the evaluation Δt , and the y-axis corresponds to the success rate. The MTS3 models were trained under different slow dynamics settings, indicated by the H values shown in the legend, using offline data with 4M transitions. Our **TAWM** employs RK4 integration method with log-uniform sampling strategy and was trained for 1.5M steps. The dashed black line indicates the default time-step size ($\Delta t = 2.5\text{ms}$). Means and 95% confidence intervals are computed over 3 seeds for TAWM, each evaluated for 10 episodes. TAWM outperforms MTS3 across different H settings, especially at large Δt values.

100% success on the Meta-World Basketball task at $\Delta t = 2.5\text{ ms}$, outperforming the baseline.

When evaluated at inference Δt values greater than the default ($\Delta t > 2.5\text{ ms}$), both TAWM (RK4) and TAWM (Euler) significantly outperform the non-time-aware baselines. We refer the readers to Figure B.5 and Figure B.6 in Figure B.6, as well as Figure B.10 in Figure B.7, for additional learning curves on other control tasks. *These results demonstrate that, despite having to learn task dynamics across different temporal scales, our time-aware model does not require additional training steps or samples to converge to a sufficiently accurate model capable of effectively solving control tasks at varying observation rates.*

Comparison to MTS3 Finally, we compare our model's performance with the Multi Time Scale World Model (MTS3) [192], a closely related approach that shares the high-level motivation of modeling world dynamics across multiple temporal scales, as introduced in Section 3.2. We would like to

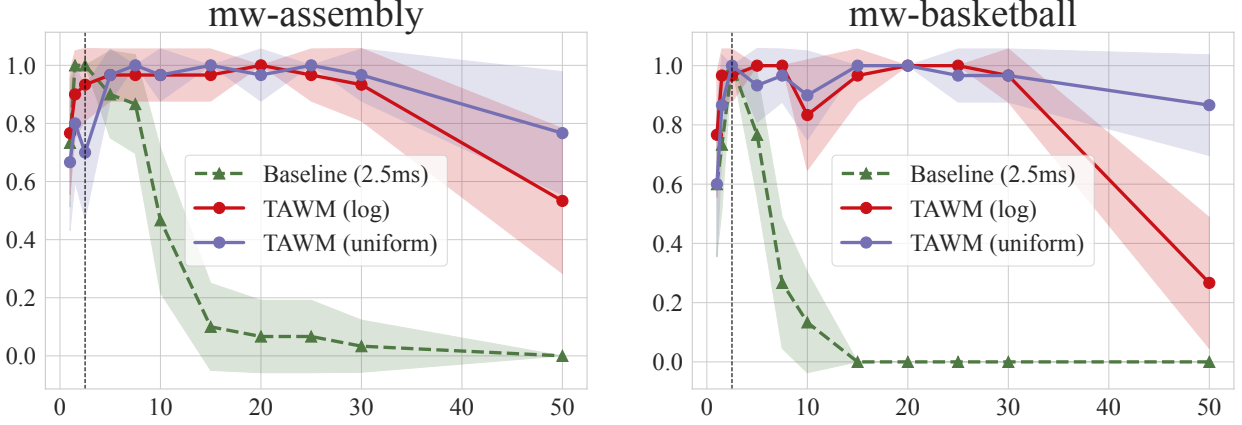


Figure 3.7: **Ablation Study on Δt Sampling Strategy.** We trained our TAWM models with **log-uniform** and **uniform** sampling strategies, both of which use the RK4 integration method. The **baseline** method was trained using only default time step ($\Delta t = 2.5ms$), which is signified with the black dashed lines. The x-axis corresponds to the evaluation Δt , and the y-axis corresponds to the success rate. Means and 95% confidence intervals are computed over 3 seeds, each evaluated for 10 episodes.

refer the readers to Appendix B.4 for detailed descriptions of MTS3’s experimental settings. In Figure 3.6, we observe that our TAWM consistently outperforms MTS3 across various settings of MTS3’s slow dynamics hyperparameter H . Notably, MTS3 exhibits rapid performance degradation as the evaluation time step increases, suggesting that it suffers from compounding errors due to long-horizon predictions. In contrast, our TAWM shows greater robustness at lower observation rates, especially on the Meta-World Faucet-Open task in which TAWM’s success rate remains at $\approx 90\%$ at $\Delta t = 50$ ms (0.05 s).

3.5.2 Ablation studies on sampling strategies

The Δt sampling strategy is a tunable hyperparameter to optimize TAWM’s performance and efficiency. In this section, we conduct an ablation study on the impact of the Δt sampling strategy during training on the performance and sample efficiency of TAWM. In Meta-World tasks, we trained the same TAWM architecture in the same Δt range using two different sampling strategies: **(1)** Log-

Uniform(0.001 s, 0.05 s) s and **(2)** Uniform(0.001 s, 0.05 s). In some tasks, such as Meta-World Assembly and Basketball, uniform sampling may outperform log-uniform sampling – particularly when the task dynamics are sufficiently slow to be captured by $\Delta t_{\max}$.

Figure 3.7 shows that while TAWM trained with the uniform sampling strategy generally performs better in most environments and achieves higher performance at low sampling rates ($\Delta t \geq 30$ ms), it exhibits lower success rates at smaller inference Δt in some environments, such as Meta-World Assembly. Additional results in other environments are presented in Figure B.1, which show a similar pattern. Regardless of the Δt sampling strategy, time-aware models consistently outperform non-time-aware baselines. Therefore, depending on the task dynamics, *TAWM can be effectively and efficiently trained with any reasonable Δt sampling strategy and is not restricted to log-uniform or uniform schemes.*

3.6 Conclusion

In this paper, we introduce a novel *time-aware world model (TAWM)* that adaptively learns task dynamics across different temporal scales. By explicitly conditioning the dynamic model on the time step size Δt and training it with a mixture of temporal scales (via Δt sampling), TAWM achieves robust performance across varying observation rates on diverse control tasks *without requiring additional training steps or samples. Empirical results show that our model consistently outperforms the non-time-aware baseline models*, which are trained only on a fixed time step size Δt for different training Δt values. We hope that the insights and results in this paper offer a new perspective on world model training, thereby contributing to the community a new, efficient, yet simple world model training method.

Limitations and Future Work. We have yet to develop a systematic methodology to analytically compute the highest frequency of the underlying task dynamics; therefore, the upper bound for the training time step size $\Delta t_{\max}$ is determined empirically. This limitation necessitates a search for $\Delta t_{\max}$ when applying TAWM to new environments (e.g., autonomous driving or PDE controllers) to improve training efficiency and convergence. An interesting direction for future work is to develop an automatic method to identify the highest frequency of task dynamics, which would determine the lower bound for sampling frequency (or the upper bound for $\Delta t_{\max}$) in training TAWM. We also plan to extend our current deterministic dynamics model to a probabilistic one to better capture real-world state transitions, particularly at larger Δt values and/or over longer horizons.

These limitations underscore that while learned world models can capture temporal structure across multiple scales, their effectiveness ultimately depends on approximations of the underlying dynamics and frequencies. This observation motivates a complementary approach that moves beyond learned models toward explicitly differentiable simulators, where system dynamics and their sensitivities can be computed analytically rather than inferred. In the next chapter, we therefore shift focus to differentiable traffic simulation, demonstrating how exact kinematic derivatives enable scalable and reliable gradient-based optimization in large-scale physical systems.

Part III

Differentiable Kinematics: Traffic

Chapter 4: Differentiable Hybrid Traffic Simulation

While the previous chapter focused on learning world models to capture system dynamics and their temporal sensitivities, an alternative and complementary approach is to compute such sensitivities directly from physically defined systems. This chapter adopts this perspective by introducing a differentiable hybrid traffic simulator, in which macroscopic and microscopic traffic dynamics are explicitly modeled and analytically differentiated across time steps. By computing exact gradients from a large-scale physical simulator rather than inferring them through learned models, this approach enables scalable and reliable gradient-based optimization for complex traffic control and planning problems.

4.1 Introduction

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 [202]. 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 [203].

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,



Figure 4.1: **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 [15].*)

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 [114]. Therefore, this approach gives us fine-grained details but often requires much more computation 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 4.1). 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 [114]. 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 paper we introduce the following main results:

1. Derivation of a differentiable macroscopic traffic model based on ARZ model [99,100] (Sec. 4.3);
2. Analytical formulation of differentiable Intelligent Driver Model [109] for agent-based vehicle dynamics (Sec. 4.4);
3. Differentiable conversion between deterministic and probabilistic instantiation of two traffic models (Sec. 4.5);
4. Application of the first differentiable hybrid traffic simulation framework to traffic control (Sec. 4.6).

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.

4.2 Previous Work

4.2.1 Traffic models

The preponderance of macroscopic models are based on the *density* and *flux* of the traffic flow; [95, 96] suggested one of the earliest models based on this idea; the LWR model is a non-linear PDE on the density of vehicles. To overcome the limitations of this model, [97, 98] added a momentum term to the LWR model, which allowed for more complex traffic dynamics than before. However, modeling traffic flow as isotropic [204, 205] led to non-physical behavior, such as negative velocity. Based on the observation that traffic flow is anisotropic, [99, 100] modified the momentum term in the previous model. The ARZ model has been used to visually simulate the traffic flow [101], and to solve traffic congestion problems [102]. Because of the basic assumption, the computational cost of these macroscopic models is proportional to the length of the simulated road, not the number of vehicles therein. Therefore, they are often more computationally efficient than the other models for large-scale environments. However, they are not suitable for simulations where fine details, such as the behavior of individual vehicles, are needed.

Microscopic models describe the motion of individual vehicles, typically tracking velocity, the bumper-to-bumper distance to its leading vehicle, and the relative velocity between them [103]. The research history dates back to [104, 105] and [106]. Modern variations of the model often include behavioral traits of individual vehicles, which make the simulation more realistic and descriptive. [107–110] are such models. These models are widely adopted in current agent-based traffic simulators, such as [206]. Since every individual vehicle observes its surrounding environments and decides its actions in these models, we can get more fine-grained details of vehicle motions than in the macroscopic mod-

els. However, their computational cost is proportional to the number of vehicles in the scene, which makes them harder to be adopted for large-scale simulations.

Hybrid models seek to combine these models in various fashions to take advantage of their complementary properties; [111–114] fall into this category and show the computational gains we can get from these models. This hybridization is commonly a spatial one, with different parts of a network running under distinct regimes. That is, these models often simulate only the regions of interest with microscopic models for accuracy and use macroscopic models elsewhere to maintain the overall flow correctness while attaining computational efficiency. This typically necessitates a technique for converting between different traffic flow representations [112]. In this work, we adopt this hybrid approach to support large-scale scenarios and let our simulator be more general so that users can select the simulation modes based on their needs.

4.2.2 Differentiable models

Differentiable models have been widely used in graphics and robotics applications like visualization [207, 208], design [209, 210], and control [211]. This paradigm enables gradient information to flow across complicated functions, and facilitates the machine learning process by improving sample efficiency [54, 212]. They often show better results in finding optimal solutions for various high-dimensional problems [213, 214].

Related to traffic simulation, there especially have been a number of differentiable simulations for a variety of systems including rigid bodies [57, 215], articulated bodies [?], soft bodies [?, ?, 86], cloth [216], and fluids [89, 217].

Because many traffic models are differentiable in nature, we can apply this technique to traffic

simulation and control. Recently, [121] has presented a differentiable agent-based traffic simulation, and showed how it can be used to solve traffic signal control problems. However, the scope of this work was limited to microscopic models for traffic, and the implementation relies on automatic differentiation, which can be computationally inefficient. In addition to traffic, this hybrid differentiable simulation framework can potentially generalize to other large-scale, multi-agent systems, such as crowds, insects, fluids, grains, etc. [218–221] as well.

4.3 Macroscopic Model

In a macroscopic traffic model, traffic evolution is described by a system of Partial Differential Equations (PDEs) on the road network.

4.3.1 Formulation

The ARZ [99, 100] model describes vehicle flow in a single lane of traffic through the following system of differential equations in one dimension:

$$\mathbf{q}_t + \mathbf{f}(\mathbf{q})_x = 0, \quad \mathbf{f}(\mathbf{q}) = \mathbf{q}u = \begin{bmatrix} \rho u \\ yu \end{bmatrix} \quad (4.1a)$$

where $\mathbf{q}(x, t) = [\rho(x, t), y(x, t)]^T$; ρ denotes the density of traffic (cars per car length), y is the relative flow of traffic, and u is the velocity of traffic. As per the notation, these terms are observed at a certain

position x at a time t , and they are related to each other as follows:

$$y(\rho, u) = \rho(u - u_{\text{eq}}(\rho)) \quad (4.1b)$$

$$u_{\text{eq}}(\rho) = u_{\text{max}}(1 - \rho^\gamma) \quad (4.1c)$$

where u_{max} denotes the maximum velocity, or speed limit, of the given lane, and $0 < \gamma < 1$ is a constant tuning parameter. Conceptually, u_{eq} represents a ‘comfortable’ velocity based on density. We used $\gamma = 0.5$ in all of the experiments.

Note that the system (4.1a) is a conservative system, where the sum of ρ and y over the entire lane is conserved, excluding the incoming and outgoing fluxes at the borders of the lane.

4.3.2 Numerical Solution

Conservation laws like the ARZ system of PDEs are typically discretized with the Finite Volume Method (FVM) [222] and integrated explicitly in time. The discretized quantities in a lane are $\mathbf{Q}_i^n$, where n refers to the time step and i refers to the i^{th} cell. The solution procedure follows:

1. Compute wave speeds λ and fluxes $\mathbf{f}$ (from Eq. (4.1a)) by solving the Riemann problem at the interface of each adjacent cell.
2. Compute global time step $\Delta t \leq \max |\lambda|$; in this work we choose a conservative, constant Δt .
3. Integrate: $\mathbf{Q}_i^{n+1} = \mathbf{Q}_i^n - \frac{\Delta t}{\Delta x} [\mathbf{f}(\mathbf{q}(b)) - \mathbf{f}(\mathbf{q}(a))]$. $\mathbf{q}(a)$ and $\mathbf{q}(b)$ are the intermediate states identified between $(\mathbf{Q}_{i-1}^n, \mathbf{Q}_i^n)$ and $(\mathbf{Q}_i^n, \mathbf{Q}_{i+1}^n)$ respectively. Δx is the length of cell i .

Details can be found in [223]; Appendix C.1.1 reproduces the six relevant cases for the ARZ Riemann problems.

4.3.3 Differentiation

Our differential technique built on this requires analytical gradients of the above; these gradients form the basis of still other gradients that are used in solving more complex traffic control problems.

In our update scheme, we compute the gradients of $\mathbf{Q}_i^{n+1}$ using $\mathbf{Q}_{i-1}^n, \mathbf{Q}_i^n$, and $\mathbf{Q}_{i+1}^n$:

$$\frac{\partial \mathbf{Q}_i^{n+1}}{\partial \mathbf{Q}_{i-1}^n} = -\frac{\Delta t}{\Delta x} \left[-\mathbf{f}'(\mathbf{q}(a)) \frac{\partial \mathbf{q}(a)}{\partial \mathbf{Q}_{i-1}^n} \right] \quad (4.1a)$$

$$\frac{\partial \mathbf{Q}_i^{n+1}}{\partial \mathbf{Q}_i^n} = \mathbf{I} - \frac{\Delta t}{\Delta x} \left[\mathbf{f}'(\mathbf{q}(b)) \frac{\partial \mathbf{q}(b)}{\partial \mathbf{Q}_i^n} - \mathbf{f}'(\mathbf{q}(a)) \frac{\partial \mathbf{q}(a)}{\partial \mathbf{Q}_i^n} \right] \quad (4.1b)$$

$$\frac{\partial \mathbf{Q}_i^{n+1}}{\partial \mathbf{Q}_{i+1}^n} = -\frac{\Delta t}{\Delta x} \left[\mathbf{f}'(\mathbf{q}(b)) \frac{\partial \mathbf{q}(b)}{\partial \mathbf{Q}_{i+1}^n} \right] \quad (4.1c)$$

We thus need the Jacobian $\mathbf{f}(\mathbf{q})$ and the partial derivatives $\frac{\partial \mathbf{q}(a)}{\partial \mathbf{Q}_{i-1}^n}, \frac{\partial \mathbf{q}(a)}{\partial \mathbf{Q}_i^n}, \frac{\partial \mathbf{q}(b)}{\partial \mathbf{Q}_i^n}, \frac{\partial \mathbf{q}(b)}{\partial \mathbf{Q}_{i+1}^n}$; this arises from differentiating solutions to Riemann problems as shown in Appendix C.1.2. The analytical gradients computed here play an important role in accelerating our traffic simulator, as shown in Section 7.5.

4.3.3.1 Time step sizes

We have derived these gradients under the assumption that Δt is constant and satisfies the CFL and stability conditions [222] for the entire simulation. It would be possible to have a dynamic Δt , since this is also differentiable (the speed λ is determined by the eigenvalues of the flux function $\mathbf{f}$). However, for the sake of the hybrid approach we ultimately use, we use a constant Δt to ensure that the microscopic simulation is stable.

4.3.3.2 Continuity issues

We have computed the gradients above assuming that the update scheme is differentiable, and in fact, we can see that it is continuous and piece-wise differentiable across most cases suggested in Appendix C.1.

We do not prove it thoroughly here, as it is generally trivial when we keep in mind that $\lim_{u_l \rightarrow u_r} \mathbf{q}_m = \mathbf{q}_l$. However, Case 1 is exceptional; there is a shock between the phase states q_l and q_m , which makes the two states discontinuous. Specifically, we cannot guarantee that q_m converges to q_l when $\lambda_s < 0$ converges to 0. Therefore, the gradients we compute when λ_s is near zero could become unstable. However, in most cases in our simulation, these cases rarely happened and did not affect the quality of our solution even when we used the possibly unstable gradients as they are; it is possible that numerical viscosity plays a role here.

4.4 Microscopic Model

In a microscopic traffic model, traffic flow is described by interactions between multiple discrete vehicles that follow certain rules. Here we use the Intelligent Driver Model (IDM) [109] to simulate such vehicles.

4.4.1 Formulation

Under the microscopic viewpoint, we can describe the state of a discrete vehicle as

$$\mathbf{q}(t) = \begin{bmatrix} p(t) \\ v(t) \end{bmatrix}, \quad (4.1)$$

where p and v denote the position and velocity of the vehicle respectively for a given time t .

Using these states to describe discrete vehicles occupying a lane, IDM describes the motion of each individual vehicle based on its relationship with the vehicle directly ahead of it; the *leading vehicle*. For the i^{th} vehicle, let $h(i)^{\text{th}}$ its leading vehicle. According to the IDM, the acceleration of the i^{th} vehicle is determined as follows:

$$\Delta p = p_{h(i)}(t) - p_i(t) - \text{length}_{h(i)} \quad (4.2a)$$

$$\Delta v = v_i(t) - v_{h(i)}(t) \quad (4.2b)$$

$$s_{\text{opt}} = s_{\text{min}} + v_i(t)T_{\text{pref}} + \frac{v_i(t)\Delta v}{2\sqrt{a_{\text{max}}a_{\text{pref}}}} \quad (4.2c)$$

$$a_i(t) = a_{\text{max}} \left[1 - \left(\frac{v_i(t)}{v_{\text{targ}}} \right)^\delta - \left(\frac{s_{\text{opt}}}{\Delta p} \right)^2 \right] \quad (4.2d)$$

There are various hyperparameters included in the model. They characterize a vehicle's motion; see Appendix C.2.1 for more details. In our experiments, we randomly initialized those hyperparameters for every single vehicle. Note that we compute s_{opt} before we compute the acceleration term. It represents the *optimal* space that the vehicle should have to avoid collision to its leading vehicle.

With the acceleration $a_i(t)$, we can update the vehicle's state using Euler's method.

4.4.2 Differentiation

Based on this update scheme, we can compute the analytical gradients of the IDM. Since the derivation is straightforward, we offer the analytical gradients in Appendix C.2.2. Note that compared to the macroscopic model, there is no significant theoretical challenge that we have to deal with in the IDM.

4.5 Macro-Micro-Macro Conversion

Our traffic simulation can transition between the continuous, flow-based macroscopic model and the discrete, agent-based microscopic model. We need to consider the differentiation across the interfaces.

4.5.1 Macro to Micro

Given a lane in the macroscopic regime that flows into a microscopic lane, we must convert the continuous representations therein to discrete vehicles at their junction. This can be done in a deterministic or stochastic fashion.

4.5.1.1 Deterministic Instantiation

Assume that $\rho(t)$ is the density of the traffic flow at the end of a macro lane on time t , and $v(t)$ is the velocity. We can set a *flux capacitor* (c.f. [223]) at the interface; this counts the total number of vehicles $N_d(t)$ that have reached this point:

$$N_d(t) = \left\lfloor \int_0^t \rho(t)v(t)dt \right\rfloor \quad (4.1)$$

where $\lfloor x \rfloor$ is the floor function that gives the greatest integer less than or equal to the real number x . $N_d(t)$ will then start from 0, and we will instantiate a vehicle when it increases by 1. The velocity of the instantiated vehicle is set to be the macroscopic traffic flow's velocity.

4.5.1.2 Stochastic Instantiation

Another possible interpretation of the macro to micro conversion is as a Poisson process. The flux of the traffic flow can be viewed as the intensity of a Poisson process. In each time step of length dt , the number of instantiated vehicles $X(t)$ follows a Poisson distribution $X(t) \sim P(\rho(t)v(t)dt)$, $P(X = k) = e^{-\rho(t)v(t)dt}(\rho(t)v(t)dt)^k k!^{-1}$. Similarly, the total number of vehicles $N_s(t)$ can be expressed as

$$N_s(t) = \int_0^t X(t) \quad (4.2)$$

As above, the velocity of the emitted discrete vehicle is the velocity of the macroscopic state at the junction of the two lanes.

4.5.2 Micro to Macro

Compared to the macro to micro conversion, agent-based information can be converted to a continuum representation more simply. The density of a cell at the interval $(l, r]$ in a macro lane can be defined as

$$\rho(l, r) = \frac{1}{r - l} \sum_{i=1}^n \mathbf{1}_{(l, r]}(p_i) \quad (4.3)$$

where $\mathbf{1}(\cdot)$ is the indicator function that identifies vehicles that lie in the interval, n is the number of vehicles and p_i is the position of i^{th} vehicle. The velocity of this cell can be the average velocities of all the vehicles therein.

4.5.3 Differentiation

We have described two types of discrete processes: instantiation of vehicles, and the indicator function. Automatic differentiation is able to compute the gradients of velocities v but cannot handle the gradients of densities ρ . To backpropagate derivatives to the densities ρ , we first create an ancillary variable $a_i = 1$ for each discrete vehicle. We rewrite Equation 4.3 as:

$$\rho(l, r) = \frac{1}{r - l} \sum_{i=1}^n \mathbf{1}_{(l, r]}(p_i) \cdot a_i \quad (4.4)$$

Since we always set $a_i = 1$, the equation constantly holds for forward simulation. But a_i can also receive the gradients for discrete vehicles from the following macro lanes. Let L be the loss function; then for each vehicle in $(l, r]$:

$$\frac{\partial L}{\partial a_i} = \frac{\partial L}{\partial \rho(l, r)} \quad (4.5)$$

The next challenge for differentiation is the vehicle instantiation process in the macro to micro conversion. We will start with the deterministic strategy: assume that the total number of vehicles reach n and $n + 1$ at time t_1 and t_2 , respectively. We therefore have the following equation

$$n + 1 = n + \int_{t_1}^{t_2} \rho(t) v(t) dt = n + a_i \quad (4.6)$$

where the i^{th} vehicle is the one instantiated at t_2 . So for $t \in (t_1, t_2]$,

$$\frac{\partial L}{\partial \rho(t)} = \frac{\partial L}{\partial a_i} v(t) dt \quad (4.7)$$

For the stochastic case, we can first compute the expectation value of the total number of vehicles according to the Poisson distribution

$$E[N_s(t)] = \int_0^t E[X(t)] = \int_0^t \rho(t)v(t)dt. \quad (4.8)$$

We notice that the expectation value of $N_s(t)$ is the same as the flux capacitor in the deterministic strategy. Similarly, we can find time t_1 and t_2 when the total number of vehicles reach n and $n+1$. For all time step $t \in (t_1, t_2]$, $\frac{\partial L}{\partial \rho(t)} = \frac{\partial L}{\partial a_i}v(t)dt$. Intuitively, if $\frac{\partial L}{\partial a_i} \leq 0$, it means the density is higher than the desired level and we need to decrease the intensity of the Poisson process. We will also validate in our numerical experiments that such estimation of gradients can effectively optimize our objective function.

4.6 Experimental Results

We show the effectiveness of our differentiable hybrid traffic simulator with application to solve a variety of traffic problems. We first show that we can accelerate the differentiable traffic simulator with the analytical gradients that we have computed above by comparing it against a baseline simulator using automatic differentiation. At the same time, we will justify the use of the hybrid model by comparing its computation time with other approaches in large-scale scenarios. Then, we prove the correctness and efficacy of our analytical gradients by solving parameter estimation problems. Lastly, we conduct experiments with traffic control problems to illustrate how we can integrate our simulator with neural networks and how we can use our simulator to solve real-world traffic problems.

We have implemented our traffic simulator with Python, and used PyTorch 1.9 [90] for automatic differentiation. All experiments were run on an Intel® Xeon® W-2255 CPU @ 3.70GHz, and traffic

rendering was mainly based on our own implementation on Unity Engine.

4.6.1 Acceleration

4.6.1.1 Analytical Gradients

Here we present how our analytical gradients can contribute to the acceleration of both forward simulation and backward propagation in our simulator. We have compared our simulator to the baseline simulator that relies on automatic differentiation to compute the gradients in the system. For all of the simulations, we have computed the gradients of the states at the last time step with respect to the states at the initial time step. For macroscopic and microscopic simulations, we used a single lane. For hybrid simulation, we used 3 lanes, which is the minimum case to cover both macro-to-micro and micro-to-macro conversion. The first and third lanes are set as macroscopic lanes and the second one is a microscopic lane. To show how the computation time changes with the scale of the simulation, we have experimented with different scales of settings.

Table 4.1 shows that our analytical gradient allows us to run both forward simulation (FW) and back propagation (BP) of the system much faster than the baseline framework. In macroscopic and hybrid simulations, the speedup was up to 4x times for forward simulation, and 18x for back propagation. In the microscopic simulation, the speedup was up to 6x times for forward simulation, and 7x times for back propagation. This shows the computational efficiency of our framework, which is necessary to implement large-scale traffic simulation.

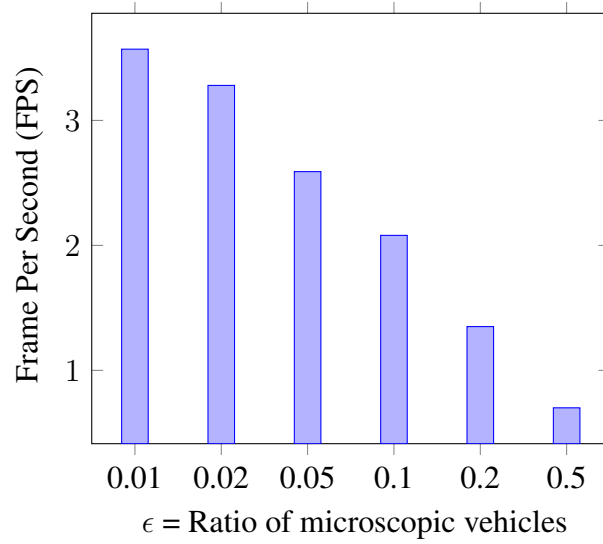


Figure 4.2: Comparison on the average frame rate (FPS) under the hybrid model, with different ϵ . The total number of vehicles is set to be 10K, and the ϵ value denotes the ratio of microscopic vehicles among them.

4.6.1.2 Hybrid Model

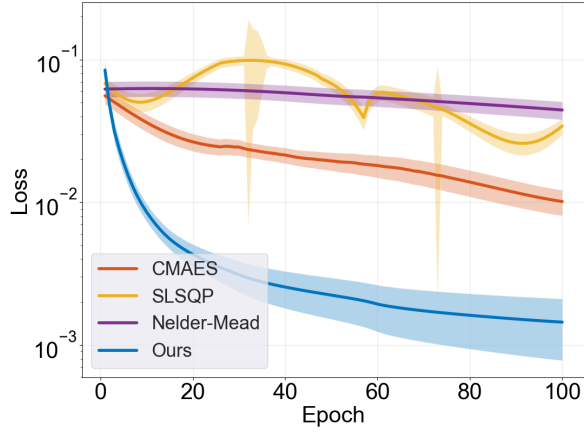
In this paper, we selected the hybrid model to run large-scale traffic simulations in interactive time, while not losing their fidelity. To justify this design choice, here we present the average frame rate for simulating a large environment, which includes approximately 10K vehicles. Note that we can specify the ratio ϵ of the vehicles which would be simulated using a microscopic model; ϵ could be regarded as an interpolant between the two models, where $\epsilon = 0$ means macroscopic and $\epsilon = 1$ means microscopic approach. We measured the average frame rate for different ϵ using single core; see Figure 4.2. Note that the average frame rates decreases as ϵ increases and thus the simulation becomes more "microscopic".

Table 4.1: Comparison of our technique against automatic differentiation for various types of simulations. Scale denotes number of time steps taken and the number of cells (for macroscopic, and hybrid)/ number of vehicles (microscopic). FW: forward simulation; BP: backpropagation. Ours outperforms autodifferentiation in both forward passes and backpropagation, by up to a factor of **18.77**, **7.84**, and **15.27** in *macroscopic*, *microscopic*, and *hybrid* traffic simulation, respectively.

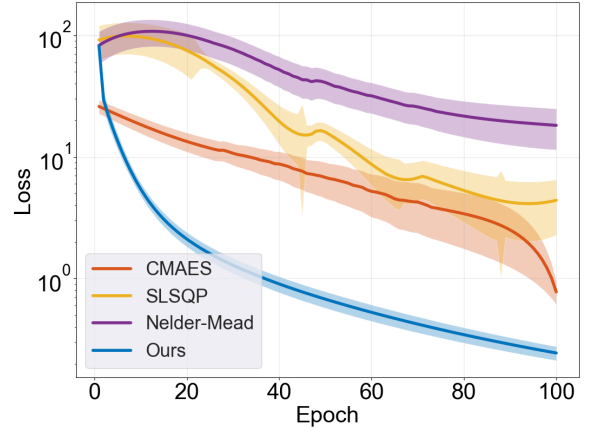
	Scale	10/1K	50/5K
Macroscopic	FW(Auto)	5.28s±0.16s	125.88s±5.66s
	FW(Ours)	1.42s±0.04s	30.06s±0.88s
	Speedup	3.71x	4.19x
	BP(Auto)	2.42s±0.20s	71.08s±6.01s
	BP(Ours)	0.18s±0.03s	3.79s±0.25s
	Speedup	13.73x	18.77x
Microscopic	FW(Auto)	1.54s±0.08s	66.74s±1.67s
	FW(Ours)	0.45s±0.02s	10.86s±0.24s
	Speedup.	3.42x	6.15x
	BP(Auto)	1.58s±0.06s	45.36s±0.95s
	BP(Ours)	0.20s±0.03s	7.10s±0.42s
	Speedup	7.84x	6.39x
Hybrid	FW(Auto)	11.81s±0.42s	310.67s±15.02s
	FW(Ours)	3.41s±0.06s	66.52s±0.47s
	Speedup	3.47x	4.67x
	BP(Auto)	2.70s±0.11s	111.06s±3.09s
	BP(Ours)	0.38s±0.02s	7.27s±0.13s
	Speedup	7.13x	15.27x

4.6.2 Parameter Estimation Problem

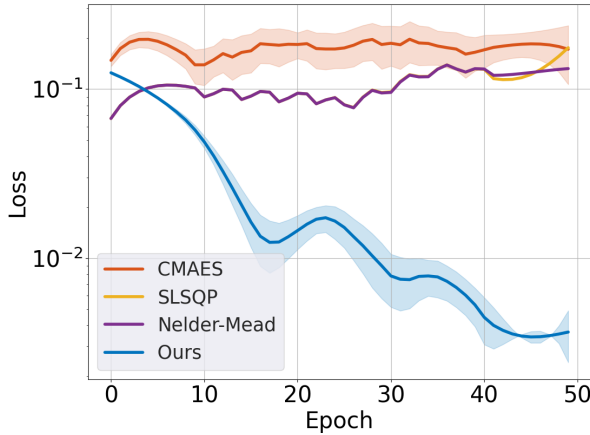
One of the most fundamental problems that we can solve with our framework is the parameter estimation problem. There are many parameters involved in our system; the speed limit is the most typical example. In our formulation, we focus on retrieving preceding states in the system based on given subsequent ones. The simplest form of this problem would be, assuming $0 \leq t \leq 1$, estimating the initial state $\mathbf{q}(0)$ given the last state $\mathbf{q}(1)$. This problem is considered to be the most basic



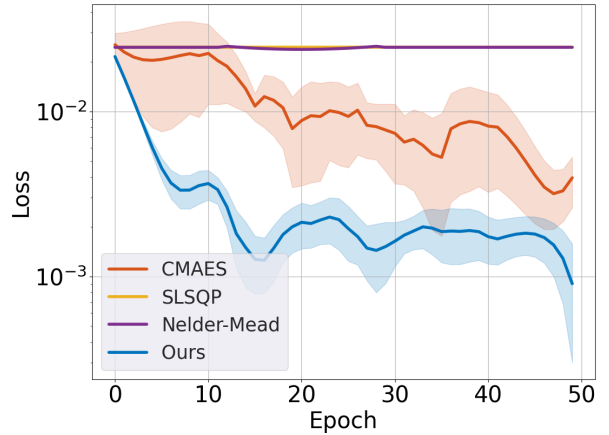
(a) Macroscopic



(b) Microscopic



(c) Hybrid (Macro-Micro-Macro)



(d) Hybrid (Micro-Macro-Micro)

Figure 4.3: Parameter estimation problem. For given final states of the system, one has to estimate the initial states that lead to the final states. (a)–(d) Our results rendered in blue exhibit much faster convergence to the correct solutions than the other state-of-the-art gradient-free optimization methods by up to *one order of magnitude*.

formulation to solve, as it only requires fundamental gradients computed analytically.

In our experiments, the simulation length was set as 10 seconds for every setting. Theoretically, we can use any simulation lengths for this experiment, but we found out that too long simulation lengths make this problem much harder than it should be and 10 seconds is a proper length to prove our approach’s efficacy. Under this setting, we have to estimate the initial traffic states that would end

up in the given final traffic states in 10 seconds. For a proposed estimate $\mathbf{q}_{\text{est}}(0)$, we can compute the estimation error with the following loss function, where n denotes the size of the state vector:

$$L = \frac{1}{n} |\mathbf{q}(1) - \mathbf{q}_{\text{est}}(1)|^2$$

We have compared our framework with other gradient-free optimization algorithms: CMA-ES [224], SLSQP [225], and Nelder-Mead [226] algorithm. Experiments were run five times with randomly initialized initial states for each of the algorithms and for each of the simulation modes. In the hybrid setting, we used 3 sequentially connected lanes. One of the experiments was done for macro-micro-macro lanes, and the other was done for micro-macro-micro lanes. The loss graph for each setting is shown in Figure 4.3. We can observe that our framework retrieves more precise initial states faster than other algorithms for every setting.

4.6.3 Traffic Control Problems

Our differentiable traffic simulator can be integrated with neural networks to enhance its ability to learn and control a given task. To test our framework’s capability, we have experimented with the “intersection signal control problem” (ITSCP) for macroscopic and hybrid simulation, and the pace car problem for microscopic simulation.

For all the experiments, we used two fully connected layers as our neural network, where each of the layers has 256 nodes. The neural network is trained to emit proper control input, which would be fed into our simulator. Then we can compute the gradient of the aggregated reward of each episode with respect to each control input therein, and apply gradient descent to directly maximize the reward. Note that this approach is same as analytical policy gradient (APG) method. Then we compare our

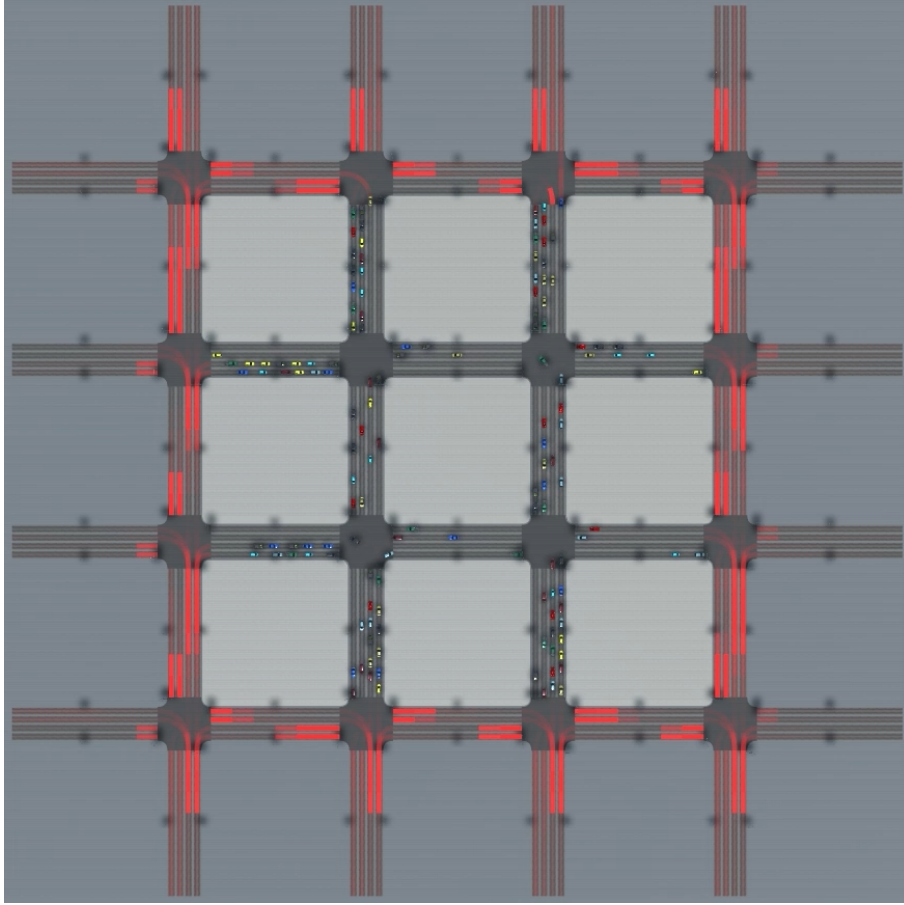


Figure 4.4: **4x4 Grid of Intersections.** Hybrid traffic simulation at a 4x4 grid of intersections: the density of each cell is rendered in red, which shows the volume of traffic agents to cross the intersection. One has to optimize time allocations for traffic lights at each intersection to maximize overall traffic flow.

results to other baseline gradient-free RL algorithms, as they are widely used to solve these kinds of control problems.

4.6.3.1 ITSCP

ITSCP is one of the most widely studied traffic control problems, because of its significance in controlling traffic congestion in urban environment [227]. We have devised a scenario that falls into this category, where an agent has to compute the optimal time allocations for traffic lights (Figure 4.4).

In this scenario, there are two different traffic lights at a given intersection: one is green across

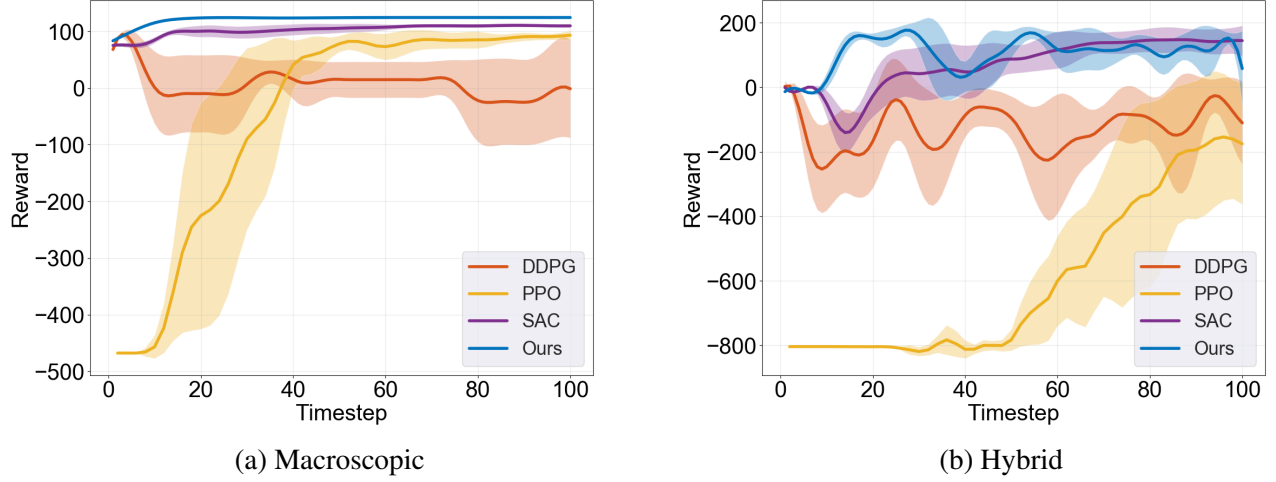


Figure 4.5: **ITSCP**. (a), (b) Learning graphs for macroscopic and hybrid simulations. Our framework shows better convergence rates and results than other RL algorithms.

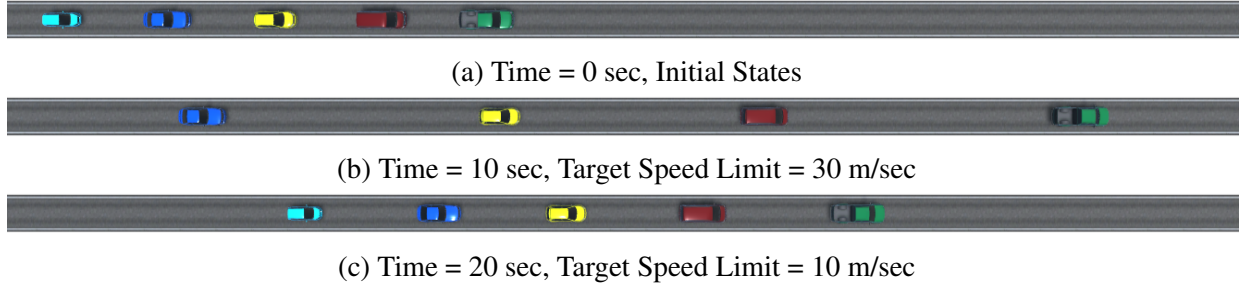


Figure 4.6: **Pace Car Problem**. Sequence of simulation images in time order, from top to bottom. (a) A pace car rendered in green guides following vehicles to maintain a target speed limit, which varies over time. It fails when it runs too fast, too slow, or goes out of the lane. (b) When a high target speed limit (30 m/sec) is given, the pace car accelerates fast, and keeps a wide distance from its following vehicles to let them run fast. (c) However, when the target speed limit is reduced (10 m/sec), it swiftly decelerates and keeps short distance from the following vehicles, to prevent them from running over the target speed limit.

the West-East direction (WE-light), and the other one is across the North-South direction (NS-light). When one of the lights is turned on, the other one has turned off automatically. In a single signal phase, the WE-light is turned on first, and then the NS-light is turned on next. There are multiple signal phases during the entire simulation, and the agent has to determine optimal time allocations for each light in each signal phase.

The input to the neural network is given as the schedule of incoming traffic flow for every

lane. Our experiments were inspired by a synthetic traffic scenario devised for another ITSCP experiment [228]. We also followed a conventional scheme to compute the reward [227]. The reward for an action is computed as the weighted sum of traffic flow that crosses the intersection (R_f) and the length of waiting queue in each lane (R_q), which is formulated as follows:

$$R = c_1 \cdot R_f + c_2 \cdot R_q.$$

For our experiments, c_1 is set as 1 and c_2 is set as -1, to maximize the traffic flow and minimize the length of waiting queue. In macroscopic environments, the length of the queue is computed as the number of cells that have speed less than certain threshold. In hybrid environments, the traffic flow is computed as the number of discrete vehicles crossing the intersection, as the central part of an intersection is simulated with a microscopic model.

For comparison, we used three baseline model-free RL algorithms; DDPG [229], PPO [36], and SAC [185]. Note that these algorithms do not use gradient information that our simulator provides. In contrast, our gradient-based optimization algorithm uses this information to optimize the objective function. Experiments were run five times for each algorithm, and Figure 4.5 shows the learning graphs of both macroscopic and hybrid environment. In the macroscopic setting, our framework converged to the near-optimal solution very quickly, while DDPG failed to learn at all and PPO and SAC learned, but not to the extent of ours. In hybrid setting, our framework and SAC both succeeded in converging to the best solution, but ours converged faster than SAC. Also, Table 4.2 shows that our best reward is better than any other algorithms for both settings.

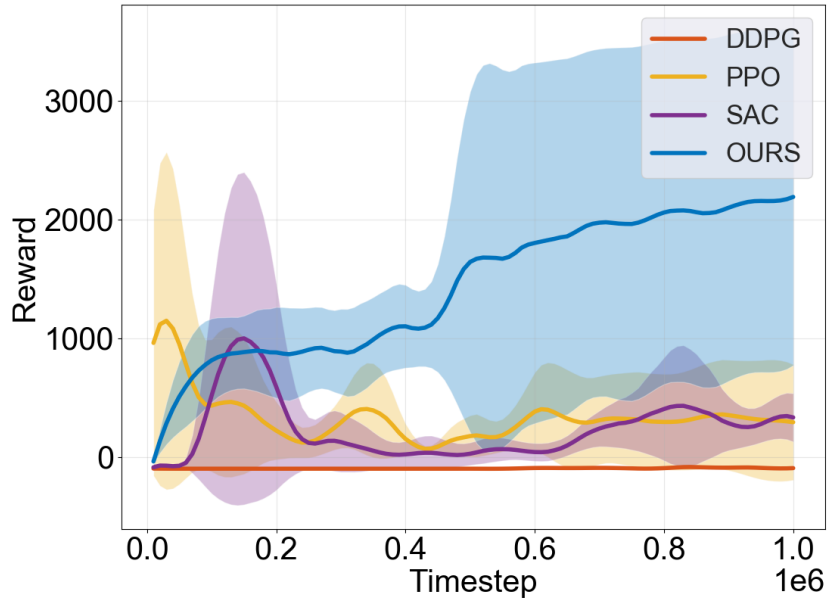


Figure 4.7: **Pace Car Learning Graph.** Our result rendered in blue shows far better performance (as indicated by higher reward values) than other algorithms, which make no significant gains in learning even after long periods.

4.6.3.2 Pace Car Problem

In this problem, we assume there is a discrete pace car, which runs in front of other vehicles that follow the IDM. The pace car leads the vehicles to enforce a specified target speed limit, which is often smaller than the lane’s speed limit (Figure 4.6).

The optimal solution for this problem is clear to the human observer; all vehicles should maintain the desired speed limit. However, the pace car has to determine both acceleration and steering at every frame to remain on the road and lead the vehicles at the same time. To introduce even more complexity, in our experiments, we used a speed limit target that varies over time. Therefore, the neural network receives other vehicle’s states and the speed limit as input, and has to compute the optimal acceleration

Table 4.2: Comparison of maximum rewards from each control problem. Our framework is able to find better solutions (indicated by higher reward values) than the other baseline, model-free RL algorithms (DDPG, PPO, or SAC) for every experiment setting (Macro, Micro, or Hybrid).

	Ours	DDPG	PPO	SAC
Macro	124.38	113.62	104.53	115.73
Micro	4482.87	4.97	3218.19	3373.99
Hybrid	208.48	174.18	158.18	191.51

and steering for next several frames, which is quite challenging. The reward is formulated as follows:

$$R = \sum_{i=1}^t \sum_{j=1}^n C_{\max} - (v_{\text{targ}} - v_{i,j})^2 \quad (4.9)$$

where $C_{\max}$ denotes an arbitrary constant value, t and n denote number of frames and number of vehicles respectively. v_{targ} and $v_{i,j}$ represent the target speed limit and j^{th} vehicle’s speed at the given frame i .

We also used three baseline RL algorithms for comparison, each of which were run 5 times. The learning graph in Figure 4.7 and the maximum rewards in Table 4.2 show that our framework achieved far better overall reward than the other algorithms. PPO and SAC also succeeded in accomplishing initial high rewards but their performance soon plunged. Our framework continuously learned and achieved better and improved results.

4.7 Conclusions

We have proposed a novel differentiable hybrid traffic simulator. In our simulator, gradients of traffic states across the time steps are computed analytically and propagated across lanes modeled under complimentary but deeply different regimes. This was made possible with our novel discrete-

continuous differentiable conversion module. Further, our analytic formulation is much more efficient than one based on automatic differentiation; therefore, our technique can offer real-time traffic simulation, as shown in Figure 4.8, that would not have been possible otherwise.

We have also shown that we can use the gradients to solve classic traffic problems. For the parameter estimation problems, our framework is able to find far better estimates than other gradient-free optimization algorithms. For control problems, our framework succeeded in finding near-optimal policies, which was not possible with other model-free RL algorithms.

4.7.1 Generalization

The concept of hybrid simulation is applicable to many different types of dynamical systems, such as fluids [230–232], crowd [233, 234], and many other multi-agent systems [235]. And, our method to differentiate the conversion between macro and micro traffic flow is applicable to a broad range of dynamical systems that are simulated by hybrid techniques, transitioning between continuous and discrete simulation domains. In general, other deterministic or stochastic instantiation process can also take advantage of our technique introduced in this paper. For example, the micro and macro representations for crowd share many similarities with traffic. The conversion between individual pedestrians and the crowd flow is analogous to the process of instantiating and removing vehicles in the roads. We also hypothesize that the algorithmic and computational framework on differentiable hybrid simulation, as proposed here, can also be extended to hybrid control of complex and autonomous systems [236, 237].



Figure 4.8: **Large-scale Hybrid Traffic simulation.** These traffic scenes consist of many hundreds of vehicles across an urban scene simulated and controlled in real time using a differentiable hybrid traffic model.

4.7.2 Limitations and Future Directions

Given these promising results, we plan to further augment the capability and application of our simulator. First, while hybrid techniques are naturally suited to metropolitan-scale scenarios, we have not yet applied this differentiable hybrid technique to these; we expect to achieve even higher performance improvements from these mega-scale experiments. We can also integrate realistic vehicle

motions, such as lane changing, to further capture complex traffic dynamics in higher fidelity. Finally, we used simulated data for solving traffic control problems in this paper, but we expect that our simulator is applicable to real-world data as well.

Beyond extending the simulator itself, an equally important question is how such differentiable physical models can be exploited for downstream optimization tasks. The availability of exact analytical gradients across large-scale traffic dynamics enables not only learning-based control, but also direct gradient-based trajectory optimization and parameter estimation. Motivated by this capability, the next chapter focuses on leveraging the differentiable traffic simulator as an optimization engine for large-scale trajectory optimization problems.

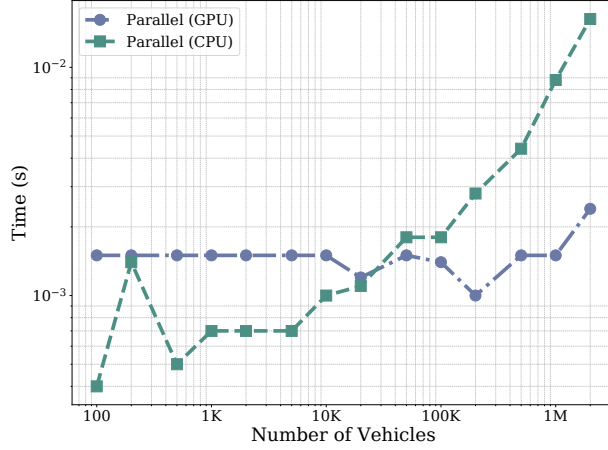
Chapter 5: Gradient-based Trajectory Optimization with Parallelized Differentiable Traffic Simulation

Building on the differentiable traffic simulator introduced in the previous chapter, this chapter focuses on directly exploiting analytical gradients for large-scale trajectory optimization. We present a parallelized, differentiable vehicle simulator based on the Intelligent Driver Model (IDM), which enables gradient-based optimization of driving behaviors and trajectories at scale. By treating the simulator itself as an optimization engine rather than a passive environment, this formulation demonstrates how physically grounded gradients can be used to filter, reconstruct, and predict traffic trajectories while preserving physical consistency.

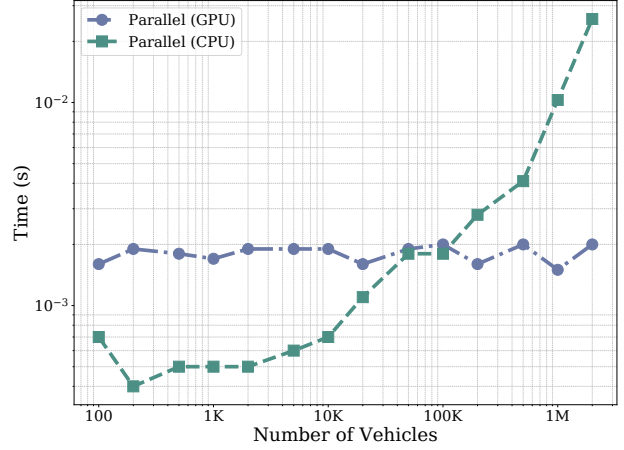
5.1 Introduction

For design, prototyping, testing, and evaluation of dynamical systems, we often rely on computer simulations that replicate reality, such as urban dynamics or natural phenomena. Traffic simulation, for example, is often used to optimize traffic light policies [238, 239] or to train autonomous vehicles in varied scenarios [116, 118]. Traffic models come in different scales, in this paper we focus on microscopic models that simulate the movement of *every individual vehicle* [104–110, 240], as opposed to macroscopic models that simulate *aggregate* traffic flow [95–100]. We adopt the Intelligent Driver Model (IDM) [109], an ODE describing car-following behavior, as it is widely used in diverse traffic simulators [115–117, 124].

When developing a traffic simulator, we aim for both efficiency and differentiability. The simulator must be efficient to gather enough data to solve the problem without excessive cost. This is



(a) Forward pass



(b) Backward pass

Figure 5.1: Computational cost of our traffic simulator. Using either (multi-threaded) CPU or GPU, our simulation takes less than 30 milliseconds *per timestep* to process 2 million vehicles in both (a) forward and (b) backward pass.

especially important as traffic problems continue to grow in scale [122, 238]. Optimization techniques are also key, and they can be classified as either *gradient-based* or *gradient-free*. Among them, gradient-free methods typically require significantly more data, because of its inferior sample efficiency. To ensure sample efficiency and reduce costs in training, testing, and evaluation, we designed our simulator to be *differentiable* via auto-differentiation frameworks.

Differentiable simulation has successfully bridged real-world dynamical systems with the power of deep learning [14, 49, 53, 55, 57, 199]. It allows a state at one timestep to be differentiable with respect to that of the previous one, enabling gradient-based optimization techniques that require fewer samples than gradient-free methods. The differentiable programming paradigm for traffic modeling has gained traction recently [121, 239, 241, 242], including data-driven approaches [125]. However, these methods often suffer from computational inefficiencies such as sequential processing and non-differentiable logic, limiting their practicality for real-time autonomous systems.

To address these limitations, we propose a *parallelized* differentiable vehicle traffic simulator

that achieves both efficiency and differentiability. Our implementation supports both CPU and GPU parallelization and can simulate up to 2 *million* vehicles in real-time (Figure 5.1). We also introduce several differentiable modifications to the original IDM to eliminate unrealistic vehicle behaviors (e.g., backward motion), which were not addressed in previous work [121, 239]. This simulator can handle various trajectory optimization tasks, including filtering, reconstruction, and prediction (Figure 5.2). Specifically, in trajectory prediction, we explicitly incorporate road network and lane membership information, which existing frameworks have not done. Our main contributions are summarized as follows:

- We present an efficient traffic simulator based on IDM, which can simulate up to 2 million vehicles in real time using either *CPU or GPU parallelization*.
- We derive and implement a *differentiable IDM* layer that is guaranteed to generate realistic vehicle motions. Using the gradients from this layer, we can leverage sample-efficient gradient-based optimization schemes for traffic optimization problems; this greatly enhances the feasibility of traffic simulation in end-to-end deep learning.
- We validate our method by solving *large-scale trajectory filtering, reconstruction, and prediction* for autonomous driving on various datasets. Our trajectories are guaranteed to follow the physical laws, while the existing methods may not always do.

5.2 Related Work

5.2.1 Traffic Simulators

Most microscopic traffic simulators are based on car-following models, which describe vehicle acceleration using driver parameters like comfortable acceleration. Notable examples include the Newell Model [106], the Gipps Model [107], the Krauss Model [240], and the Intelligent Driver Model (IDM) [109]. Popular simulators such as SUMO [115], HighwayEnv [116], CARLA [117], and FLOW [118] use IDM to model vehicle movements, while others like MATSIM [119] and VISSIM [120] employ similar car-following models. IDM’s widespread use in simulators influenced our choice. However, these IDM-based simulators lack parallelization and differentiability, making them less suited for solving large-scale traffic problems.

Recent advancements in machine learning and GPU hardware have led to the development of parallelized or differentiable traffic simulators for large-scale traffic simulation and urban mobility optimization. MOSS [122] uses GPU power for city-scale traffic simulation based on IDM and MOBIL [123], but it lacks differentiability, preventing application with gradient-based optimization and deep learning. Other approaches have applied differentiable methods to traffic optimization and control [121, 239, 241], but they are not as efficient as MOSS. Data-driven simulators like Waymax [124] and GPUDrive [125] offer both GPU acceleration and differentiation but focus on replaying real-world trajectory logs from the Waymo Open Dataset (WOMD) [126], without reactive traffic behaviors. To the best of our knowledge, our simulator is the *first model-based traffic simulator that achieves both high efficiency and differentiability for optimization, learning, and control* (Table D.1).

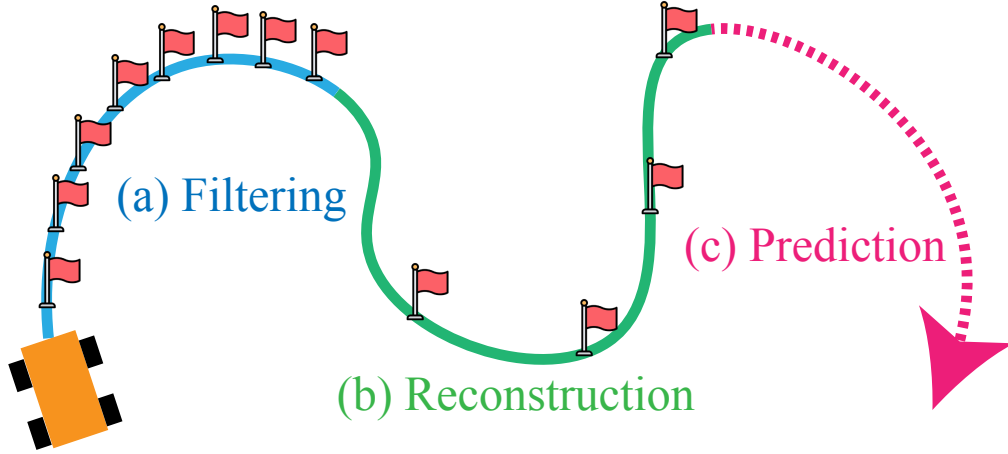


Figure 5.2: **Trajectory Optimization Problems.** For the given data points (red flags), we can fit a simulated trajectory to them by optimizing IDM variables. (a) When data points are dense, we can filter physically inaccurate noises in the original trajectory. (b) When they are sparse, we can reconstruct dense trajectories. (c) We can even predict future trajectories based on the IDM variables.

5.2.2 Trajectory Optimization

We address three trajectory optimization problems: trajectory filtering, reconstruction, and prediction (Figure 5.2). For trajectory filtering, we assume we have dense observations (e.g., vehicle positions) that can be simply concatenated to construct a dense trajectory. However, numerical derivation on this trajectory amplifies noise, leading to physically impossible estimates of speed or acceleration [243] (Figure 5.4). To address this, various filtering methods have been proposed, including moving averages [244–246], spline smoothing [247], and wavelet analysis [243]. While this has been studied under “trajectory reconstruction,” we use the term “trajectory filtering” to differentiate it from our reconstruction task, which focuses on generating dense trajectories from sparse data points – essentially an interpolation task. Unlike previous model-free approaches, *our model-based approach is guaranteed to generate physically plausible trajectories.*

In the trajectory prediction task, many data-driven methods forecast future trajectories based on

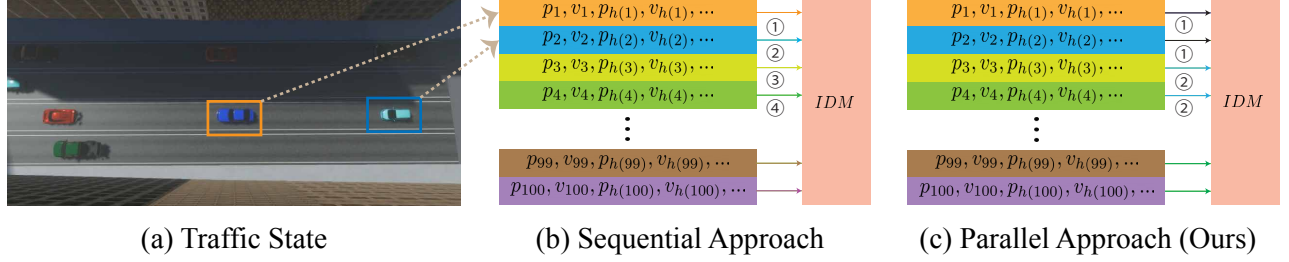


Figure 5.3: **Overall Framework.** In our traffic simulator, for each frame, (a) we first collect variables for each vehicle in the scene to use in IDM. These variables include the position (p_i) and velocity (v_i) of each vehicle, and its relationship to its leading vehicle ($p_{h(i)}$, $v_{h(i)}$). Since we can apply IDM to each vehicle’s variables independently, we can process them in parallel, rather than sequentially. In (b) and (c), the process ordering is depicted with colors and numbers. In (c), we assume there are two computational units that can run in parallel.

historical data and road context [248–253]. Current state-of-the-art methods use transformer architectures to model trajectories as sequences of positions, similar to natural language processing. However, none incorporate explicit micro-simulated dynamic states into the modeling process, instead relying on highly parameterized models to implicitly capture traffic dynamics during training. This gap exists due to the lack of fast, differentiable traffic simulators—most are either sequential or non-differentiable. While some differentiable environments are used in reinforcement learning tasks [199, 239], no such simulators are available for imitative or open-loop methods.

5.3 Method

5.3.1 Preliminary

Our traffic simulator is built on the Intelligent Driver Model (IDM) [109], which assumes that a vehicle’s acceleration is determined solely by its relationship to the vehicle directly ahead in the same lane. Specifically, if the position and speed of the i -th vehicle at time t are denoted as $p_i(t)$ and $v_i(t)$, and the index of its leading vehicle is $h(i)$, we compute Δp (distance gap) and Δv (speed difference)

as follows:

$$\Delta p = p_{h(i)}(t) - p_i(t) - length_{h(i)}$$

$$\Delta v = v_i(t) - v_{h(i)}(t).$$

Based on these variables, we can compute the optimal spacing s_{opt} and acceleration $a_i(t)$ of the i -th vehicle as

$$s_{\text{opt}} = s_{\text{min}} + v_i(t)T_{\text{pref}} + \frac{v_i(t)\Delta v}{2\sqrt{a_{\text{max}}a_{\text{pref}}}} \quad (5.1)$$

$$a_i(t) = a_{\text{max}} \left[1 - \left(\frac{v_i(t)}{v_{\text{targ}}} \right)^\delta - \left(\frac{s_{\text{opt}}}{\Delta p} \right)^2 \right]. \quad (5.2)$$

The formulation for IDM above includes several hyperparameters that describe the vehicle's behavioral traits: a_{max} and a_{pref} for maximum and preferred acceleration, s_{min} for the minimum distance gap, T_{pref} for the preferred time to maintain current speed, and v_{targ} for the target speed. In this paper, we use meters (m) for position, meters per second (m/s) for speed, and meters per second squared (m/s²) for acceleration.

After we obtain the acceleration of each vehicle at time t , we compute its position and speed at the next time step, $t + \Delta t$ using Euler integration as follows:

$$\begin{aligned} p_i(t + \Delta t) &= p_i(t) + \Delta t \cdot v_i(t), \\ v_i(t + \Delta t) &= v_i(t) + \Delta t \cdot a_i(t). \end{aligned} \quad (5.3)$$

The time step Δt can be adjusted depending on the scenario. While a differentiable version of this formulation has been proposed in several prior works [121, 239], these implementations often lack

parallelization and result in unrealistic behaviors, such as negative speed. In the following sections, we demonstrate how to address these limitations.

5.3.2 Parallelization Scheme

Figure 5.3 illustrates the overall framework of our simulator. We first collect the necessary per-vehicle information for each simulation frame to compute acceleration using IDM. As shown in Figure 5.3(b) and (c), each vehicle’s data can be represented as independent data blocks. Due to this independence, instead of processing the blocks sequentially (Figure 5.3(b)), we process them in parallel (Figure 5.3(c)). Additionally, the IDM kernel is implemented in a differentiable manner. With this parallelization, our simulator can simulate up to 2 million vehicles in real-time on either CPU or GPU and efficiently compute gradients (Figure 5.1).

5.3.3 Unrealistic Behaviors

We observed that IDM often estimates negative accelerations that are physically impossible or end up with negative speed. To prevent these cases, we first set the lower bound of s_{opt} to 0, as it should be a positive value by its conceptual definition (‘optimal spacing’). Then, we set the lower bound of $a_i(t)$ to $a_{lb} = \max(-v_i(t)/\Delta t, a_{min})$, where a_{min} is a new hyperparameter that corresponds to the (physically valid) maximum deceleration. Note that we can guarantee that the next speed estimated from this acceleration term is non-negative, as we use Euler integration (Eq. 5.3). We choose to use Euler integration, as opposed to other time integration methods such as Runge-Kutta, for its simplicity. While other methods may be more numerically stable, we find that Euler works well with a sufficiently small timestep δt and minimal computing cost.

To implement these lower-bound operations in a differentiable way, we use the softplus function as follows:

$$s_{opt}^* = \log(1 + \exp(s_{opt}))$$

$$a_i(t)^* = a_{lb} + \log(1 + \exp(a_i(t) - a_{lb})).$$

Note that $a_i(t)^*$ is guaranteed to exist in $[a_{min}, a_{max}]$.

5.4 Application

In this section, we provide our solutions to the trajectory optimization problems using our differentiable simulator.

5.4.1 Trajectory Filtering & Reconstruction

Since the trajectory filtering and reconstruction tasks differ only in data point density, we apply the same solution to both. We assume that each trajectory is represented by a sequence of timestamps and corresponding vehicle positions in a 1-dimensional space. For the i -th trajectory, let $\mathbb{T}_j^i$ (sec) be the j -th timestamp and $\mathbb{P}_j^i$ (m) the corresponding vehicle position. Our goal is to generate dense trajectories that interpolate these points while adhering to physical laws. Simple linear interpolation often leads to sudden changes in estimated speed and acceleration, which are physically infeasible (Figure 5.4).

Therefore, we use our traffic simulator to reconstruct physically consistent trajectories that adhere to the given data as much as possible. To be specific, if we denote the temporal length of an

i -th trajectory as $\mathbb{T}_l^i$, we simulate the motion of the ego vehicle from time 0 to $\mathbb{T}_l^i$, using δt as Δt in Eq. 5.3. Then, for each time stamp $\mathbb{T}_j^i$ in the input, we can find the nearest time step $k_j^i \cdot \delta t$, where k_j^i is a positive integer, and compute the reconstruction loss for the i -th trajectory based on mean absolute error (MAE):

$$L_{recon}^i = \sum_{j=1}^l |\mathbb{P}_j^i - \mathbf{P}[k_j^i]|, \quad (5.4)$$

where $\mathbf{P}$ is an array that stores the position of the vehicle during simulation. As our simulator supports large-scale simulation, we simulate all trajectories at once and compute the final loss $L = \sum_{i=1}^N L_{recon}^i$.

Once the reconstruction loss is computed, we minimize it using gradient-based optimization techniques. We optimize five IDM parameters $(a_{max}, a_{pref}, T_{pref}, s_{min}, v_{targ})$, which represent driver behavior and remain fixed for each trajectory. These parameters are initialized to 10, 2, 1, 5, and 50, respectively, with constraints of $[5, 10]$, $[0.1, 5]$, $[0.1, 5]$, $[1, 10]$, $[20, 60]$. We also optimize the lists of Δp_k and Δv_k for each simulation step k , initializing them to 10 and 0. Lastly, a_{min} is set to -10 and remains fixed during optimization.

5.4.2 Trajectory Prediction

In trajectory prediction, the goal is to forecast a future trajectory based on a historical trajectory and scene context. Recent open-loop deep learning methods use neural networks to identify patterns from real-world trajectory data and predict human-like outcomes [248, 249, 253–256]. However, common failure modes early in training involve scenarios where vehicles collide with the leading vehicle or go beyond road boundaries. Additionally, deep neural network approaches for trajectory prediction are known to be data inefficient, despite achieving SOTA results [257].

We use our differentiable traffic simulation to provide a *learning-free* performance baseline using

differentiable ODE modeling and inferred road context. First, we infer traffic states from vehicle trajectories and lane shapes. For our experiments, we use the Waymo Open Motion Dataset (WOMD) [126]. Agents are assigned to lanes based on proximity to lane polyline features, and both lane polylines and agent states are projected into 1-dimensional representations for our simulator. We then fit IDM parameters using the historical trajectory, similar to the optimization process in the trajectory reconstruction task. In deep learning approaches, this information would be encoded into context embeddings. Finally, we simulate trajectories in the 1-dimensional space using the fitted IDM parameters and project them back onto the original lane polylines.

We hypothesize that by using this approach, the “predicted” trajectory will closely align with real-world trajectories, without requiring deep neural networks for learning. If successful, these ODE-based rollouts could provide a strong baseline and lower bound on performance metrics for more complex, learning-based methods.

This task is not feasible with existing simulators. While some simulators use ODE-based traffic state construction, their lack of differentiability or parallelization makes them impractical for large-scale trajectory prediction. Conversely, large-scale parallelized simulators do not explicitly construct traffic states from road data or use ODEs to model agents, making them heavily dependent on learned policies and real-world data logs for future rollouts.

5.5 Experimental Results

In this section, we provide experimental results. First, we present the experimental results about the efficiency of our simulator. Then, we provide results of trajectory filtering and reconstruction tasks together and discuss those of trajectory prediction in the end. Our approach is implemented in Python

Table 5.1: **Quantitative comparison for trajectory filtering & reconstruction task.** For each task, we compare the average positional error rate (Pos.), magnitude of estimated acceleration (Acc.), physically implausible trajectory rate (Imp.), and average computation time per trajectory (Time.) between compared methods. For estimated acceleration magnitude (m/s^2), we provide mean and standard variation (μ/σ).

Task	Value	Linear	MA	EMA	Spline	Ours
Fil.	Pos.↓	0.00%	0.01%	0.02%	0.00%	0.08%
	Acc.↓	6.4/5.5	0.9/0.9	0.5/0.6	3.5/2.1	0.5/0.5
	Imp.↓	100%	8.18%	4.61%	2.00%	0%
	Time.↓	0.115s	0.233s	0.237s	23.9s	0.308s
Rec.	Pos.	0.00%	0.05%	0.06%	0.01%	0.13%
	Acc.	1.8/5.4	0.5/0.7	0.2/0.4	1.6/1.9	0.3/1.1
	Imp.	99.84%	9.83%	7.24%	4.12%	0%
	Time.	0.033s	0.038s	0.042s	3.37s	0.164s

and PyTorch [90]. All of our experiments were done on a workstation equipped with Intel® Xeon® W-2255 CPU @ 3.70GHz and a single NVIDIA RTX A5000 GPU.

5.5.1 Computational Efficiency

To assess the computational efficiency of our simulator, we measured its running time for both the forward and backward passes. The total time was divided by the number of simulation steps, as running time scales proportionally with the number of steps. To evaluate scalability and device impact, we tested different numbers of vehicles using either CPU (with 16 threads) or GPU. As shown in Figure 5.1, our simulator can handle up to 2 million vehicles in real-time on both CPU and GPU. The CPU performs efficiently due to the use of 16 threads. For smaller numbers of vehicles, the CPU is faster since launching GPU jobs introduces overhead. However, the GPU demonstrates better scalability as the number of vehicles goes beyond $20K$.

5.5.2 Trajectory Filtering & Reconstruction

5.5.2.1 Dataset

For the trajectory filtering task, we used the publicly available Next Generation Simulation (NGSIM) dataset [258], specifically 6,101 trajectories from US Highway 101, recorded every 100 milliseconds, providing dense data for filtering. For trajectory reconstruction, we used a non-public dataset containing 21,750 sparse trajectories from Singapore, recorded with a minimum sampling interval of 1 second. Each trajectory is up to 20 minutes long.

5.5.2.2 Baselines

For comparison, we used two interpolation methods (linear, spline) and two filtering techniques (moving average [244, 245], exponential moving average [246]) as baselines. For interpolation, we set δt to 0.1 and 1.0 for each task, estimating vehicle positions at every δt to generate dense trajectories. This matches our method, where δt is the simulation time step. For the filtering task, linear interpolation results were identical to the raw data, as the data was recorded *every* 100 milliseconds. For spline interpolation, we fitted a cubic B-spline curve using Scipy’s optimization function [259], setting bounds on the first and second derivatives (speed and acceleration) to $[0, \infty)$ and $[-10, 10]$, respectively, to ensure realistic trajectories as much as possible. For filtering, we applied both methods to the dense trajectories from linear interpolation, using a window size of 9 for moving average (MA) and a smoothing width of 5 for exponential moving average (EMA), following [245] and [246].

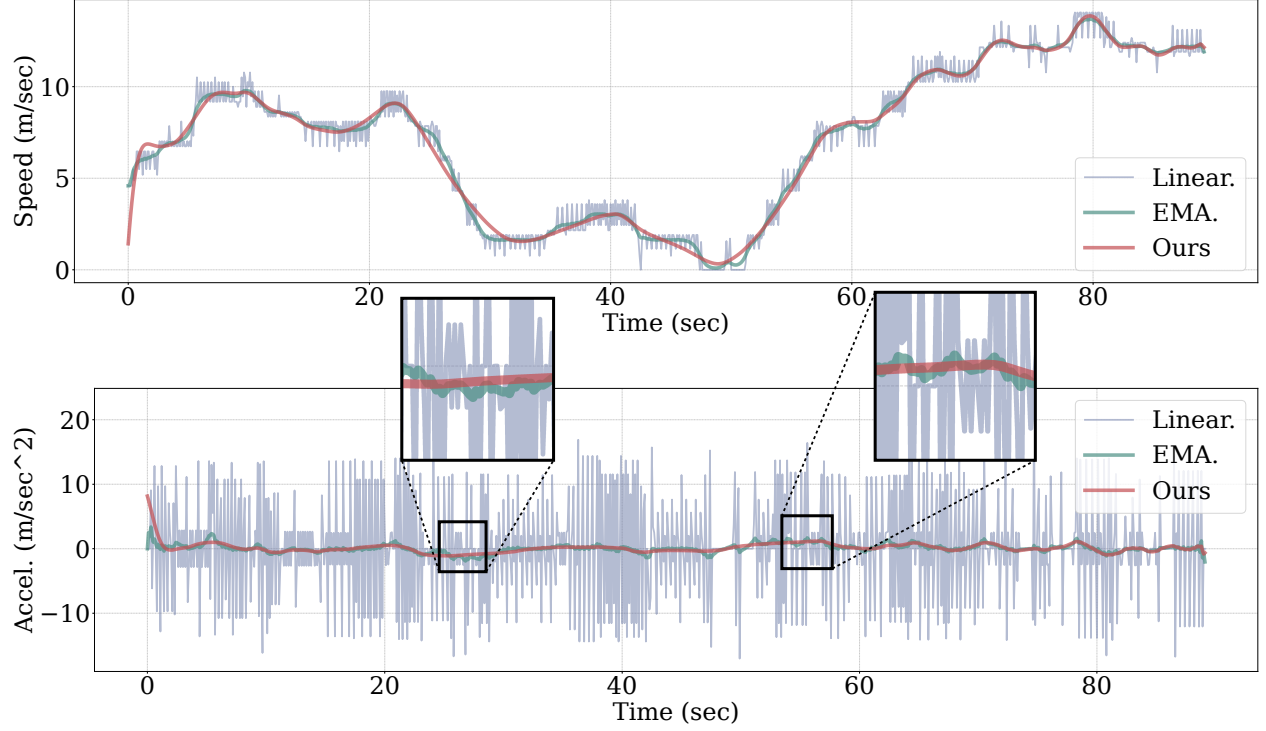


Figure 5.4: **Qualitative comparison for trajectory filtering task.** For comparison, we use the speed (up) and acceleration (down) profiles for a single trajectory in the NGSIM dataset. Estimated acceleration from our method exhibits a more stable pattern than the baseline methods.

5.5.2.3 Optimization Details

In our optimization process, we used the Adam optimizer [260] with 500 optimization steps for both tasks. During optimization, we linearly reduced the learning rate from an initial value of 0.1 to 0.01. The initial position $p(0)$ and speed $v(0)$ for each trajectory were set to 0 and $(\Delta\mathbb{P})/\Delta t$, where $\Delta\mathbb{P}$ is the distance between the first two data points.

5.5.2.4 Quantitative Comparison

Table 5.1 presents quantitative results based on four criteria:

1. **Average positional error rate (Pos.):** Represents the precision of the trajectory, calculated by finding the nearest time step in the estimated dense trajectory for each data point, measuring the distance, dividing by the total trajectory length, and averaging across all points.
2. **Statistics about estimated accelerations (Acc.):** Mean and standard deviation of absolute accelerations. Lower values usually indicate more stable trajectories.
3. **Ratio of physically implausible trajectories (Imp.):** Ratio of trajectories with any step where absolute acceleration exceeds 10, as most real-world absolute accelerations stay below 3.
4. **Running Time (Time.):** Running time per trajectory.

As we can see in Table 5.1, linear and spline interpolation achieved the best results in positional error, but most of their trajectories were physically invalid. MA and EMA also performed well in positional error and gave impressive results on the second criterion. However, they still produced physically invalid trajectories, and they are not negligible—one of the trajectories exhibited an acceleration of 66, which is physically impossible. In contrast, *trajectories from our method were all physically valid, and showed stable acceleration pattern, with only a slight loss in positional error.* In terms of computational cost, our method was up to 5 times slower than simple linear interpolation *but processed each trajectory in under 0.4 seconds with parallelization.*

5.5.2.5 Qualitative Comparison

To qualitatively assess the optimized trajectories, we plot the speed and acceleration profile of a trajectory from the NGSIM dataset in Figure 5.4, comparing our results with linear interpolation and EMA. Linear interpolation simply mirrors the raw data (possibly with noise), so the resulting speed and

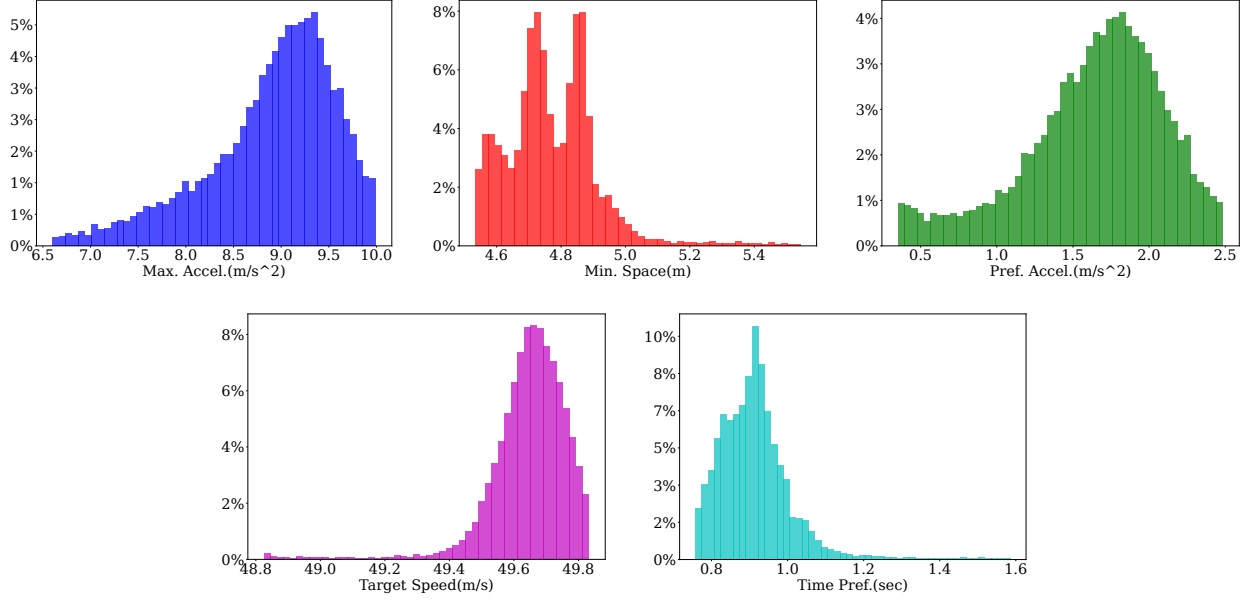


Figure 5.5: **Distribution of optimized IDM parameters in trajectory reconstruction task.** Note that the distribution of each parameter adheres well to the general, real-world scenarios.

acceleration profiles contain significant physically invalid noise. After applying the EMA filter, high-frequency noise is reduced, and both profiles become smoother – however, it is still not guaranteed to be physically valid. Our method produces even smoother results, and they are guaranteed to be physically valid.

5.5.2.6 IDM Parameters

As mentioned earlier, we optimize five IDM parameters during trajectory optimization. To assess the impact of optimizing these variables, we reran the trajectory reconstruction task with fixed IDM parameters. As expected, this produced worse results than our original approach: the positional error rate increased to 0.22%, and the estimated acceleration became (0.5/1.7). Optimizing the IDM parameters allows for better results by providing more flexibility. Figure 5.5 shows the distribution of the optimized IDM parameters, which closely aligns with real-world observations. This demonstrates

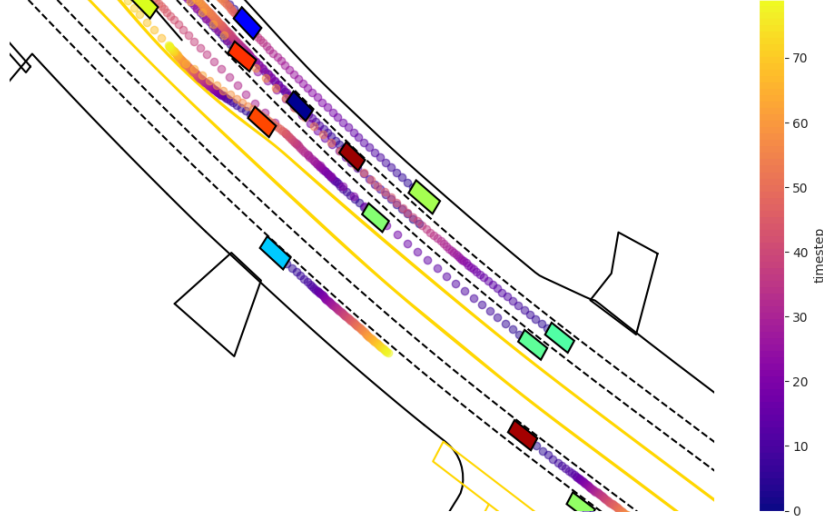


Figure 5.6: **Training-free future rollouts on the Waymo Open Motion Dataset (WOMD).** We show qualitative results in trajectory forecasting with our differentiable traffic simulator. In the trajectory forecasting task, we infer the traffic state based on road graph and agent observations, including agent membership and order within lanes. Then, we fit IDM parameters for each vehicle to its 1-second historical trajectory and roll out trajectories via IDM for all agents according to their current lane. As shown above, agent vehicles maintain realistic trajectories with no deep learning or ground truth.

that, in addition to producing better results, *our method can infer driver behavior during optimization*, offering an advantage over baseline methods.

5.5.3 A Training-Free Trajectory Forecasting Baseline

In trajectory forecasting experiments, we use our differentiable simulator to fit IDM parameters and perform training-free trajectory rollouts on the Waymo Open Motion Dataset (WOMD) [126] validation set, which contains 15.5K 20-second segments split into overlapping 9-second samples. Each sample represents a traffic scenario centered on an autonomous vehicle. Using the Adam optimizer, we fit IDM parameters to 1-second trajectory histories sampled at 10Hz over 10 timesteps. To compute the traffic state as positions along 1-dimensional lanes, we project each vehicle’s historical trajectory onto lane centers, where only vehicles within a certain distance threshold of lane centers are considered for

future prediction. After solving for IDM parameters with gradient-based optimization, we initialize the traffic state to the starting state and roll out 8-second futures at 10 Hz, for a total of 80 frames.

For each vehicle active in traffic lanes, we evaluate their performance with WOMD benchmark metrics. That is, we show results in mean average precision (mAP), minimum average displacement error (minADE), minimum final displacement error (minFDE), and miss rate (MissRate):

1. **mAP**: Mean average precision across all trajectory types and agent classes, where trajectory types include straight, straight-left, straight-right, left, right, left u-turn, right u-turn, and stationary.
2. **minADE**: L2 norm between closest prediction and ground truth.
3. **minFDE**: L2 norm between closest final position and ground truth final position.
4. **Miss Rate**: % of agent predictions outside of a threshold from ground truth.

Closer details on metrics can be found in Ettinger et al.’s work introducing the benchmark [126].

We show quantitative results on WOMD validation set in Table 5.2, and compare to recent SOTA models in trajectory forecasting [251, 252, 254–256, 261]. In this result, we consider all benchmark trajectories and agent classes in the validation set, including that of cyclists and pedestrians in addition to vehicles.

Expectedly, results with our traffic simulator does not outperform SOTA methods, and should not be directly compared against models trained on ground truth annotations. However, *we find that our simulator achieves comparable mAP performance (0.354) and acceptable distance metric performance (7.862 minADE), given no training on a large amount of pre-annotated data, where the next best mAP performance is Multipath++ [253] at 0.393*. We point out that minADE and minFDE

Table 5.2: **Waymo Open Motion Dataset validation set metric comparisons between SOTA models and training-free rollouts generated by our differentiable simulator.** We show metric performance results generated from our simulator versus current SOTA models for trajectory prediction. Without any neural networks or training, future predictions using our simulator can attain 0.3541 mAP on validation—while this result does not outperform SOTA models, this performance is attained with *no training*, where IDM parameters are only fitted on **1-second of data**. Missing values are not reported by the respective original work.

Method	mAP↑	minADE↓	minFDE↓	MissRate↓	Params
Multipath++ [253]	0.393	0.978	2.305	0.440	-
MTR [252]	0.416	0.605	1.225	0.137	66M
ControlMTR [255]	0.422	0.590	1.201	0.132	66M
Wayformer_Ens [256]	0.430	0.530	-	-	60M
MTR++ [251]	0.435	0.591	1.199	0.130	86M
EDA [261]	0.435	0.571	1.173	0.119	-
MTRv3_Ens [254]	0.488	0.554	1.104	0.110	-
Ours	0.354	7.862	15.668	0.586	5

values are high perhaps due to poor performance on samples involving non-straight driving. In this experiment, vehicles merely advance forward along the current road centers. mAP values may be competitive purely due to overrepresentation of straight driving in benchmark datasets. Still, our method provides an interesting blind driving benchmark for trajectory forecasting work, which does not involve any deep learning whatsoever. We visualize an example rollout on a WOMD validation sample in Figure 5.6, where we show qualitatively that simulated rollouts are realistic and follow road structures.

5.6 Conclusion

In this paper, we introduced a parallelized differentiable traffic simulator based on IDM, capable of simulating up to 2 million vehicles in real-time on parallel CPU or GPU implementations. To address physically implausible results, we modified the original IDM and demonstrated the simulator’s effectiveness in solving trajectory optimization problems. For trajectory filtering and reconstruction,

our method reliably estimated physically plausible trajectories. Additionally, we showed that differentiable traffic simulation serves as a strong training-free baseline for trajectory forecasting and paves the way for integrating traffic simulation with deep learning for robust, traffic-aware techniques.

A limitation of our work is its focus on IDM-based vehicle modeling, which could be extended to include ODE-based pedestrian and cyclist behavior for more robust agent representations. Future work could also explore more complex loss functions for improved trajectory accuracy and support for complex road networks. Nonetheless, our simulator can serve as a core computational layer for integrating with traffic simulators that handle such systems.

More broadly, this chapter demonstrates how gradient-informed optimization can be effectively applied to large-scale, physically grounded dynamical systems when accurate sensitivities are available. Importantly, the underlying principle—leveraging gradients as measures of input–output sensitivity—extends beyond time-evolving traffic dynamics to problems defined by spatial structure and geometric relationships. Motivated by this observation, the next part of the thesis shifts focus from differentiable dynamics to differentiable geometry, exploring how similar ideas can be applied to traditionally discrete mesh representations.

Part IV

Differentiable Geometry: Mesh

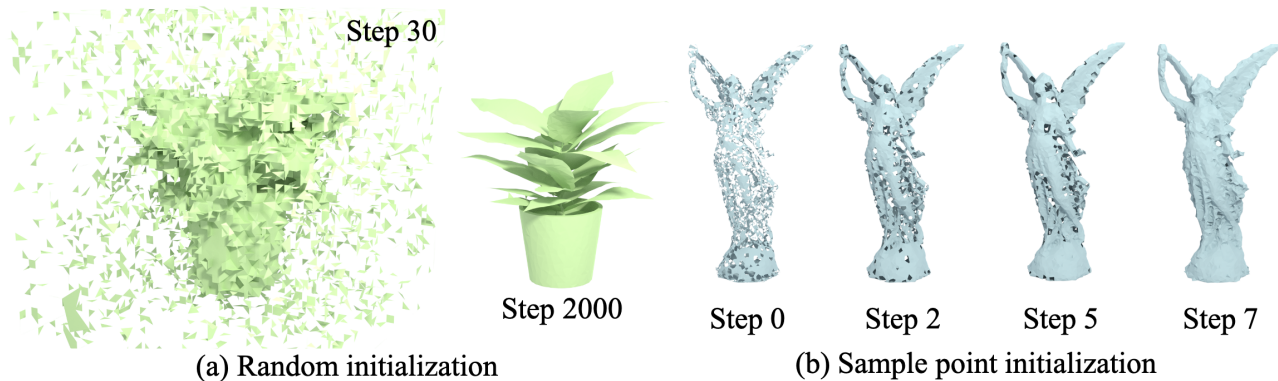


Figure 6.1: ($\rightarrow$) **Optimization process.** We can optimize our mesh starting from either (a) random state or (b) initialization based on sample points for faster convergence. Mesh connectivity changes dynamically during the optimization. To make this topology change possible, we compute existence probability for an arbitrary set of faces in a differentiable manner.

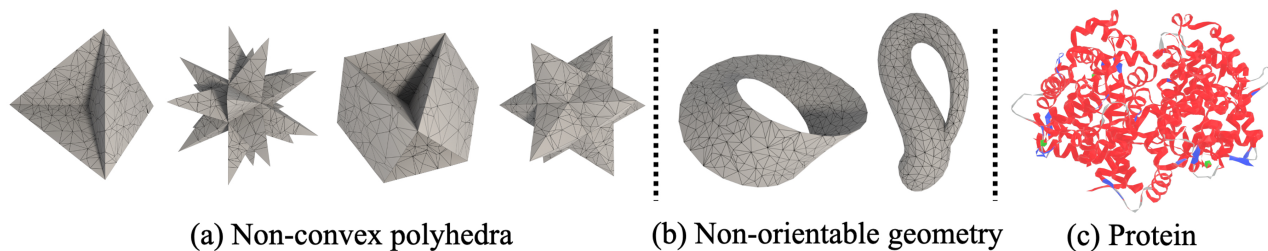


Figure 6.2: **Versatility of DMesh.** DMesh can represent diverse geometry in differentiable manner, including (a) non-convex polyhedra of different Euler characteristics, (b) non-orientable geometries (Möbius strip, Klein bottle), and (c) complex protein structure (colored for aesthetic purpose).

The previous chapters demonstrated how gradient-informed optimization can be effectively applied to time-evolving dynamical systems, from policy learning to large-scale physical simulation and trajectory optimization. This chapter extends the same underlying principle beyond temporal dynamics to static but highly structured geometric representations. We introduce **DMesh**, a differentiable mesh representation that reformulates discrete geometric connectivity and topology so that

gradients—interpreted as sensitivities between geometric parameters and observations—can be computed and exploited for optimization.

6.1 Introduction

Polygonal meshes are widely used in modeling and animation due to their diverse, compact and explicit configuration. Recent AI progress has spurred efforts to integrate mesh generation into machine learning, but challenges like varying topology hinder suitable differentiable mesh representations. This limitation leads to reliance on differentiable intermediates like implicit functions, and subsequent iso-surface extraction for mesh creation [9–12, 262, 263]. However, meshes generated by such approaches can be misaligned at sharp regions and unnecessarily dense [11], not suitable for down-stream applications that require light-weight meshes. This limitation necessitates us to develop a truly differentiable mesh representation, not the intermediate forms.

The fundamental challenge in creating a differentiable mesh representation lies in formulating both the vertices’ geometric features and their connectivity, defined as edges and faces, in a differentiable way. Given a vertex set, predicting their connectivity in a free-form way using existing machine learning data-structures can cost significant amount of computation and be difficult to avoid irregular and intersecting faces. Consequently, most studies on differentiable meshes simplify the task by using a mesh with a pre-determined topology and modifying it through various operations [3, 4, 159, 264]. This work, on the contrary, ambitiously aims to establish a general 3D mesh representation, named as DMesh, where both mesh topology and geometric features (e.g. encoded in vertex location) can be simultaneously optimized through gradient-based techniques.

Our core insight is to use differentiable Weighted Delaunay Triangulation (WDT) to divide a

convex domain, akin to amber encapsulating a surface mesh, into tetrahedra to form a mesh. To create a mesh with arbitrary topology, we select only a subset of triangular faces from the tetrahedra, termed the “real part”, as our final mesh. The other faces, the “imaginary part”, support the real part but are not part of the final mesh (Figure 6.4). We introduce a method to assess the probability of a face being part of the mesh based on weighted points that carry positional and inclusiveness information. Optimization is then focused on the points’ features to generate the triangular mesh. The probability determination allows us to compute geometric losses and rendering losses during gradient-based optimization that optimizes **connectivity and positioning**.

The key contributions of our work can be summarized as follows.

- We present a novel differentiable mesh representation, **DMesh**, which is versatile to accommodate various types of mesh (Figure 6.2). The generated meshes can represent shapes more effectively, with much less number of vertices and faces (Table 6.2).
- We propose a computationally efficient approach to differentiable WDT, which produces robust probability estimations. While exhaustive approach [13] requires quadratic computational cost, our method runs in approximately *linear time*.
- We provide efficient algorithms for reconstructing surfaces from both point clouds and multi-view images using DMesh as an intermediate representation.
- We finally propose an effective regularization term which can be used for mesh simplification and enhancing triangle quality.

Additionally, to further accelerate the algorithm, we implemented our main algorithm and differentiable renderer in CUDA, which is made available for further research.

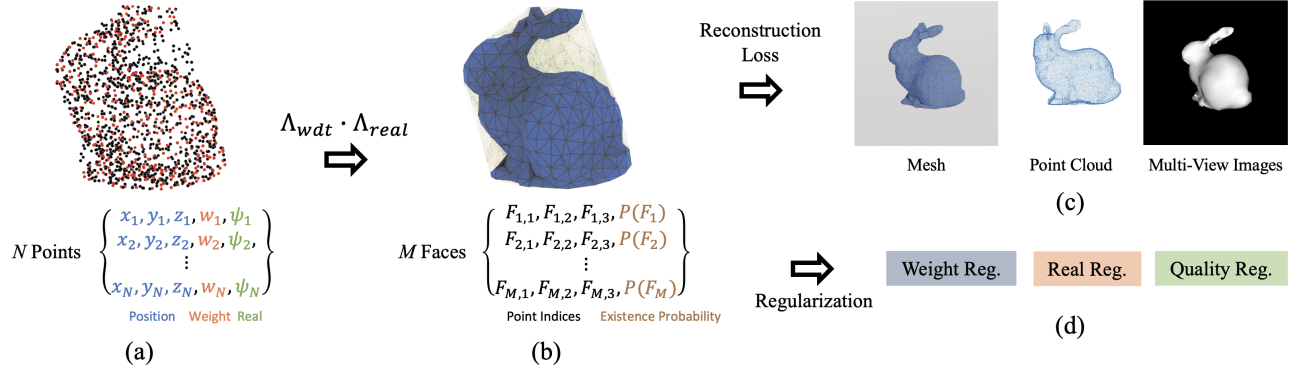


Figure 6.3: Our overall framework to optimize mesh according to the given observations. **(a)**: Each point is defined by a 5-dimensional feature vector: position, weight, and real value. Points with larger real values are rendered in red. **(b)**: Given a set of points, we gather possibly existing faces in the mesh and evaluate their probability in differentiable manner. **(c)**: We can compute reconstruction loss based on given observations, such as mesh, point cloud, or multi-view images. **(d)**: To facilitate the optimization process and enhance the mesh quality, we can use additional regularizations.

6.2 Related Work

6.2.1 Shape Representations for Optimization

Recently, using neural implicit functions for shape representation gained popularity in graphics and vision applications [5, 127–132]. They mainly use volume density, inspired by [127], to represent a shape. However, because of its *limited accuracy* in 3D surface representation, neural signed distance functions (SDFs) [5, 6, 133, 134] or unsigned distance functions (UDFs) [7, 8] are often preferred. After optimization, one can recover meshes using iso-surface extraction techniques [22, 138].

Differing from neural representations, another class of methods directly produce meshes and optimize them. However, they assume that the overall mesh topology is fixed [4, 24, 160, 161], only allowing local connectivity changes through remeshing [3]. Learning-based approaches like BSP-Net [265] allow topological variation, but their meshing process is not differentiable. Recently, differentiable iso-surface extraction techniques have been developed, resulting in high-quality geometry

reconstruction of various topologies [9–12, 18, 136, 262]. Unfortunately, meshes relying on iso-surface extraction algorithms [22, 138] often result in unnecessarily dense meshes that could contain geometric errors. In contrast, **our approach addresses these issues: we explicitly define faces and their existence probabilities, and devise regularizations that yield simplified, but accurate meshes based on them** (Table 6.2). See Table E.1 for more detailed comparisons to these other methods.

6.2.2 Delaunay Triangulation for Geometry Processing

Delaunay Triangulation (DT) [266] has been proven to be useful for reconstructing shapes from unorganized point sets. It’s been shown that DT of dense samples on a smooth 2D curve includes the curve within its edges [267, 268]. This idea of using DT to approximate shape has been successfully extended to 3D, to reconstruct three-dimensional shapes [269] for point sets that satisfy certain constraints. However, these approaches are deterministic. Our method can be considered as a differentiable version of these approaches, which admits gradient-based optimization.

More recently, [13] focused on this DT’s property to connect points and tessellate the domain, and proposed a differentiable WDT algorithm to compute smooth inclusion, namely existence score of 2-simplexes (triangles) in 2 dimensional space. However, it is not suitable to apply this approach to our 3D case, as there are computational challenges (Section 6.3.2). Other related work, VoroMesh [17], also used Voronoi diagrams in point cloud reconstruction, but their formulation cannot represent open surfaces and is only confined to handle point clouds.

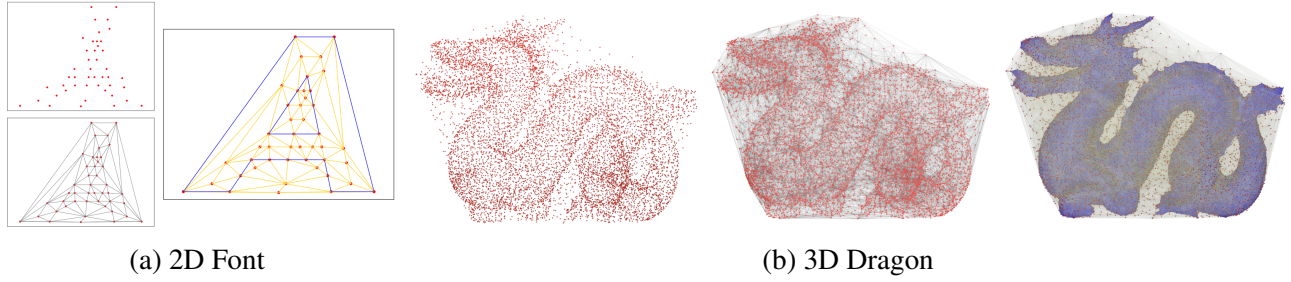


Figure 6.4: Illustration of our mesh representation for 2D and 3D cases. **(a)**: Our representation in 2D for a letter “A”. **(b)**: Our representation in 3D for a dragon model. Blue faces are “*real part*” and yellow ones are “*imaginary part*”.

6.3 Preliminary

6.3.1 Probabilistic Approach to Mesh Connectivity

To define a traditional, non-differentiable mesh, we specify the vertices and their connectivity. This connectivity is discrete, meaning for any given triplet of vertices, we check if they form a face in the mesh, returning 1 if they do and 0 otherwise. To overcome this discreteness, we propose a **probabilistic** approach to create a fully differentiable mesh – given a triplet of vertices, we evaluate the probability of a face existing. This formulation enables differentiability not only of **vertex positions** but also of their **connectivity**.

Note that we need a procedure that tells us the existence probability of any given face to realize this probabilistic approach. This procedure must be **1)** differentiable, **2)** computationally efficient, and **3)** maintain desirable mesh properties, such as avoiding non-manifoldness and self-intersections, when determining the face probabilities.

Among many possible options, we use *Weighted Delaunay Triangulation (WDT)* (Figure 6.6(a)) and a point-wise feature called the “real” value (ψ) to define our procedure. Each vertex in our frame-

work is represented as a 5-dimensional¹ vector including position (3), WDT weight (1), and real value (1) (Figure 6.3(a)). Given the precomputed WDT based on vertices' positions and weights, we check the face existence probability of each possible triplets. Specifically, **1**) a face F must exist in the WDT, and then **2**) satisfy a condition on the real values of its vertices to exist on the actual surface. We describe the probability functions for these conditions as Λ_{wdt} and Λ_{real} :

$$\Lambda_{wdt}(F) = P(F \in \text{WDT}), \quad \Lambda_{real}(F) = P(F \in \text{Mesh} \mid F \in \text{WDT}). \quad (6.1)$$

Then we get the final existence probability function, which can be used in downstream applications (Figure 6.3), as follows:

$$\Lambda(F) = P(F \in \text{Mesh}) = \Lambda_{wdt}(F) \cdot \Lambda_{real}(F). \quad (6.2)$$

This formulation attains one nice property in determining the final mesh – that is, it prohibits self-intersections between faces. When it comes to the other two criteria about this procedure, Λ_{wdt} function's differentiability and efficiency is crucial, as we design Λ_{real} to be a very efficient differentiable function based on real values (Section 6.4.2). Thus, we first introduce how we can evaluate Λ_{wdt} in a differentiable and efficient manner, which is one of our main contributions.

6.3.2 Basic Principles

¹In 2D case, a vertex is a 4-dimensional vector including position (2), WDT weight (1), and real value (1).

To begin with, we use (d, k) pair to denote a k -simplex (Δ^k) in d -dimensional space. For a 3D mesh, we observe that our face F corresponds to $(d = 3, k = 2)$ in Figure 6.5(b). To compute the probability Λ_{wdt} for the face, we use power diagram (PD), which is the dual structure of WDT (Figure 6.6(a)). While previously Rakotosaona et al. (2021) proposed a differentiable 2D triangulation method for the $(d = k = 2)$ case (Figure 6.5(c)), it suffers from quadratically increasing computational cost (e.g. it takes 4.3 seconds to process

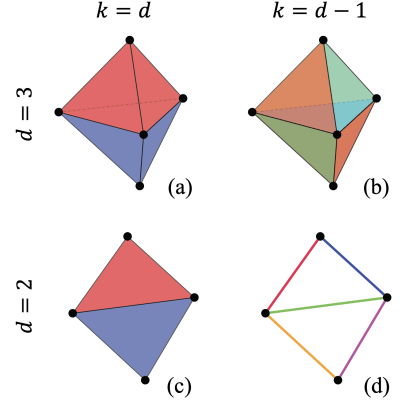


Figure 6.5: Renderings of Δ^k_s for different pairs of (d, k) . Different Δ^k_s are rendered in different colors.

10K points in 2D, Table E.2) and unreliable estimation when $(k < d)$. We will discuss later how our formulation conquer these computational challenges. Our setting for 3D meshes are similar to 2D meshes, the $(d = 2, k = 1)$ case in Figure 6.5(d), where a triangular face reduces to a line. Therefore, we will mainly use this setting to describe and visualize basic concepts for simplicity. However, note that it can be generalized to any (d, k) case without much problem. We generalize the basic principles suggested by [13] to address our cases. For formal definitions of the concepts in this section, please refer to Appendix E.2.

Let S be a finite set of points in $\mathbb{R}^{d=2}$ with weights. For a given point $p \in S$, we denote its weight as w_p . We call those weighted points in S as “vertices”, to distinguish them from general unweighted points in $\mathbb{R}^2$. Then, we adopt power distance $\pi(p, q) = d(p, q)^2 - w_p - w_q$ as the distance measure between two vertices in S . As depicted in Figure 6.6, $\Delta^{k=1}$ is a 1-simplex, which is a line connecting two vertices p_i and p_j in S . Its dual form D_{Δ^1} (red line) is the set of unweighted points² in $\mathbb{R}^2$ that are located at the same power distance to p_i and p_j . In the power diagram, the power cell C_p (gray cell) of a vertex p is the set of unweighted points that are closer to p than to any other vertices in S . We can

²We treat unweighted points’ weight as 0 when computing the power distance.

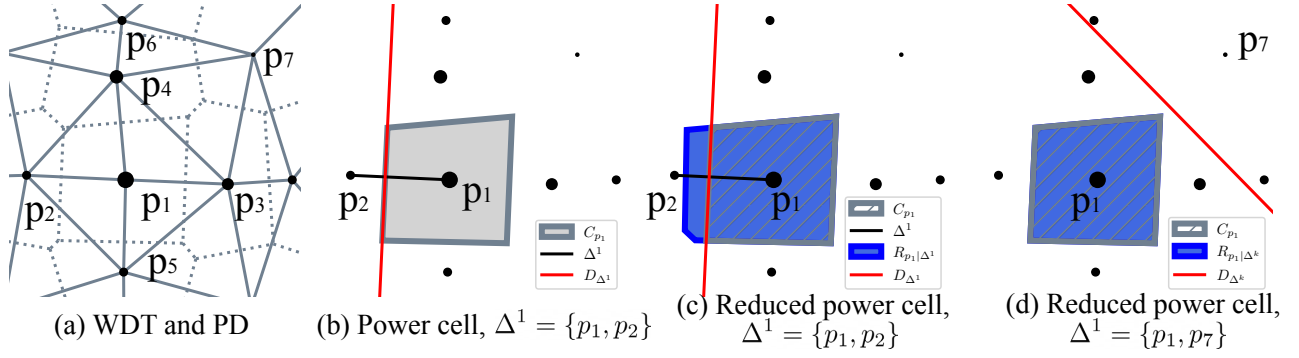


Figure 6.6: **Basic concepts to compute existence probability of given 1-simplex (Δ^1) when $d = 2$.** (a): WDT and PD of given set of weighted vertices are rendered in solid and dotted lines. The size of a vertex represents its weight. (b): Power cell of p_1 (C_{p_1}) is rendered in grey. Also, Δ^1 is rendered in black line, of which dual line (D_{Δ^1}) is rendered in red. (c), (d): For given Δ^1 , reduced power cell of p_1 for the Δ^1 ($R_{p_1|\Delta^1}$) is rendered in blue, with the original power cell (grey). We can evaluate the existence of Δ^1 in WDT by computing the signed distance from D_{Δ^1} to $R_{p_1|\Delta^1}$.

use C_p and D_{Δ^1} to measure the existence of Δ^1 . From Figure 6.6, we can see that when $\Delta^1 = \{p_i, p_j\}$ exists in WDT, its dual line D_{Δ^1} aligns exactly with C_{p_i} 's boundary, while when $\Delta^1 = \{p_i, p_j\}$ doesn't exist in WDT, D_{Δ^1} is outside C_{p_i} .

To make this measurement less binary when Δ^1 exists, we use the expanded version of power cell called "reduced" power cell ($R_{p|\Delta}$), introduced by [13]. The reduced power cell of $p_i \in S$ for $\Delta^1 = \{p_i, p_j\}$ is computed by excluding p_j from S when constructing the power cell³. For example, when Δ^1 exists, $R_{p_i|\Delta^1}$ will expand towards p_j 's direction (Figure 6.6(c)), and D_{Δ^1} will "go through" $R_{p_i|\Delta^1}$. In contrast, when Δ doesn't exist, even though we have removed p_j , $R_{p_i|\Delta^1}$ will not expand (Figure 6.6(d)), and thus D_{Δ^1} stays outside of $R_{p_i|\Delta^1}$.

Now we can define a signed distance field $\tau_{pt}(x, R)$ for a given reduced power cell R , where the signed distance is measured as the distance from the point $x \in \mathbb{R}^d$ to the boundary of the reduced power cell (sign is positive when inside). Then, we can induce the signed distance between a dual

³In 3D case where $k = 2$ and $\Delta^2 = \{p_i, p_j, p_k\}$, p_j and p_k would be ignored for $R_{p_i|\Delta^2}$.

form D and a reduced power cell R :

$$\tau(D, R) = \max_{x \in D} \tau_{pt}(x, R). \quad (6.3)$$

As illustrate in Figure 6.6(c) and (d), $\tau(D_{\Delta^1}, R_{p_1|\Delta^1})$ is positive when Δ^1 exists, while negative when it does not exist in WDT.

This observation can be generalized as follows:

Remark 6.3.1 Δ^k exists in WDT if and only if $\forall p_i \in \Delta^k, \tau(D_{\Delta^k}, R_{p_i|\Delta^k}) > 0$.

In fact, the sign of every $\tau(D_{\Delta^k}, R_{p_i|\Delta^k})$ is same for every $p_i \in \Delta^k$. Therefore, we can use its average to measure the existence probability of Δ_k , along with sigmoid function σ :

$$\Lambda_{wdt}(\Delta^k) = \frac{1}{k+1} \sum_{p \in \{\Delta^k\}} \sigma(\tau(D_{\Delta^k}, R_{p|\Delta^k}) \cdot \alpha_{wdt}), \quad (6.4)$$

where α_{wdt} is a constant value used for the sigmoid function. $\Lambda_{wdt}(\Delta^k)$ is greater than 0.5 when Δ^k exists, aligning with our probabilistic viewpoint and being differentiable.

Computational Challenges. As mentioned before, Rakotosaona et al. (2021) solved the problem for the case where $(d = k = 2)$ where the dual D_{Δ^k} is a single point. Naïvely applying their approach for computing Eq. 6.3 to our cases poses two computational challenges:

- **Precision:** When $(d = 3, k = 2)$ or $(d = 2, k = 1)$, D_{Δ^k} becomes a **line**, not a single point.

Finding a point on the line that maximizes Eq. 6.3 is not straightforward.

- **Efficiency:** When naïvely estimating the reduced power cell in exhaustive manner, the computational cost increases with the number of points at a rate of $O(N^2)$, where N is the number of

points.

See Appendix E.2.2 for a detailed discussion of these limitations.

6.4 Formulation

6.4.1 Practical approach to compute Λ_{wdt}

We introduce a practical approach to resolve the two aforementioned challenges. Specifically, we propose constructing the PD first and use it for computing lower bound of Eq. 6.3 in an efficient way. We also propose to handle the two cases separately: whether Δ^k exists in the WDT or not.

First, when the simplex Δ^k does not exist, we choose to use the negative distance between the dual form and the normal power cell, $-d(D_{\Delta^k}, C_p)$. This is the lower bound of Eq. 6.3:

$$-d(D_{\Delta^k}, C_p) \leq \tau(D_{\Delta^k}, R_{p|\Delta^k}) < 0, \quad (6.5)$$

as $C_p \subset R_{p|\Delta^k}$. See Figure 6.6(d) for this case in $(d = 2, k = 1)$ case. We can observe that computing this distance is computationally efficient, because the normal PD only needs to be computed once for all in advance. Moreover, C_p is a convex polyhedron, and D_{Δ^k} is a (convex) line, which allows us to find the distance between line segments on the boundary of C_p ⁴ and D_{Δ^k} , and choose the minimum.

Second, we analyze the case when the simplex Δ^k exists in WDT. In this case, we have to construct the reduced power cell $R_{p|\Delta^k}$ for given Δ^k , which requires much additional cost. Instead of doing it, we leverage pre-computed PD to approximate the reduced power cell. Then, we pick a point

⁴This holds when $d = 3$. When $d = 2$, we can use vertices of C_p .

$v \in D_{\Delta^k} \cap R_{p|\Delta^k}$, where $p \in \{\Delta^k\}$, and the following holds:

$$0 \leq \tau_{pt}(v, R_{p|\Delta^k}) \leq \tau(D_{\Delta^k}, R_{p|\Delta^k}), \quad (6.6)$$

because $v \in R_{p|\Delta^k}$ and by the definition at Eq. 6.3. In our case, since $D_{\Delta^k} \cap R_{p|\Delta^k}$ is a line segment, we choose its middle point as v to tighten this lower bound. See Figure 6.6(c) for the line segment in $(d = 2, k = 1)$ case. We use this bound when Δ^k exists. Note that computing this lower bound is also computationally efficient, because we can simply project v to the reduced power cell.

To sum up, we can rewrite Eq. 6.3 as follows.

$$\tau(D_{\Delta^k}, R_{p|\Delta^k}) = \begin{cases} \tau_{pt}(v, R_{p|\Delta^k}) & \text{if } \Delta^k \in \text{WDT} \\ -d(D_{\Delta^k}, C_p) & \text{else} \end{cases} \quad (6.7)$$

By using this relaxation, we can get lower bound of Eq. 6.3, which is reliable because it always has the same sign. Also, we can reduce the computational cost from $O(N^2)$ to nearly $O(N)$, which is prerequisite for representing meshes that have more than $1K$ vertices in general. See Appendix E.2.2 for the computational speed and accuracy of our method, compared to the previous one. Finally, we implemented our algorithm for computing Eq. 6.7 in CUDA for further acceleration.

6.4.2 Definition of Λ_{real}

Λ_{real} evaluates the existence probability of a k -simplex Δ^k in our mesh when it exists in WDT.

To define it, we leverage per-point value $\psi \in [0, 1]$. To be specific, we compute the minimum ψ of the points in Δ^k in differentiable way: $\Lambda_{real}(\Delta^k) = \sum_{p \in \Delta^k} \kappa(p) \cdot \Psi(p)$, where $\kappa(p) = e^{-\beta \cdot \Psi(p)} / \sum_{q \in \Delta^k} e^{-\beta \cdot \Psi(q)}$, and Ψ is function that maps a point p to its ψ value. We set $\beta = 100$.

Along with Λ_{wdt} that we discussed before, now we can evaluate the final existence probability of faces in Eq. 6.2. We also note here, that when we extract the final mesh, we only select the faces of which Λ_{wdt} and Λ_{real} are larger than 0.5.

6.4.3 Loss Functions

DMesh can be reconstructed from various inputs, such as normal meshes, point clouds, and multi-view images. With its per-vertex features and per-face existence probabilities $\Lambda(F)$, we can optimize it with various reconstruction losses and regularization terms. Please see details in Appendix E.3.

6.4.3.1 Reconstruction Loss (L_{recon})

First, if we have a normal mesh with vertices $\mathbb{P}$ and faces $\mathbb{F}$, and we want to represent it with DMesh, we should compute the additional two per-vertex attributes, WDT weights and real values. We optimize them by maximizing $\Lambda(\mathbb{F})$ since these faces lie on the reference mesh. Conversely, for the remaining set of faces $\bar{\mathbb{F}}$ that can be defined on $\mathbb{P}$, we should minimize $\Lambda(\bar{\mathbb{F}})$. Together, they define the reconstruction loss for mesh input (Appendix E.3.1).

For reconstruction from point clouds or multi-view images, we need to optimize for all features including positions. For point clouds, we define our loss using Chamfer Distance (CD) and compute the expected CD using our face probabilities (Appendix E.3.2). For multi-view images, we define the loss as the L_1 loss between the given images and the rendering of DMesh, interpreting face probabilities as face opacities. We implemented efficient differentiable renderers to allow gradients to flow across face opacities (Appendix E.3.3).

6.4.3.2 Regularizations

Being fully differentiable for both vertices and faces, DMesh allows us to develop various regularizations to improve the optimization process and enhance the final mesh quality. The first is **weight regularization** (L_{weight}), applied to the

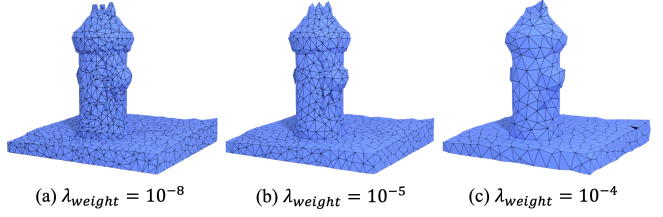


Figure 6.7: **Results with different λ_{weight} .**

dual Power Diagram of the WDT (Appendix E.3.4). This regularization reduces the structural complexity of the WDT, controlling the final mesh complexity (Figure 6.7). The next is **real regularization** (L_{real}), which enforces nearby points to have similar real values and increases the real values of points adjacent to high real value points (Appendix E.3.5). This helps remove holes or inner structures and makes faces near the current surface more likely to be considered (Appendix E.4). The final regularization, **quality regularization** (L_{qual}), aims to improve the quality of triangle faces by minimizing the average expected aspect ratio of the faces, thus removing thin triangles (Appendix E.3.6).

To sum up, our final loss function can be written as follows:

$$L = L_{recon} + \lambda_{weight} \cdot L_{weight} + \lambda_{real} \cdot L_{real} + \lambda_{qual} \cdot L_{qual},$$

where λ values are hyperparameters. In Appendix E.5, we provide values for these hyperparameters for every experiment. Also, in Appendix E.5.3, we present ablation studies for these regularizations.

6.5 Experiments and Applications

In this section, we provide experimental results to demonstrate the efficacy of our approach. First, we optimize vertex attributes to restore a given ground truth mesh, directly proving the differentiability of our design. Next, we conduct experiments on 3D reconstruction from point clouds and multi-view images, showcasing how our differentiable formulation can be used in downstream applications.

For the mesh reconstruction problem, we used three models from the Stanford 3D Scanning Repository [270]. For point cloud and multi-view reconstruction tasks, we used four closed-surface models from the Thingi32 dataset, four open-surface models from the DeepFashion3D dataset, and three additional models with both closed and open surfaces from the Objaverse dataset and Adobe Stock. These models are categorized as "closed," "open," and "mixed" in this section. Additionally, we use nonconvex polyhedra of various Euler characteristics and non-orientable geometries to prove our method’s versatility.

We implemented our main algorithm for computing face existence probabilities and differentiable renderer used for multi-view image reconstruction in CUDA [271]. Since we need to compute WDT before running the CUDA algorithm, we used WDT implementation of CGAL [272]. We implemented the rest of logic with Pytorch [167]. All of the experiments were run on a system with AMD EPYC 7R32 CPU and Nvidia A10 GPU.

6.5.1 Mesh to DMesh

In this experiment, we demonstrate that we can preserve most of the faces in the original normal triangular mesh after converting it to DMesh using the mesh reconstruction loss introduced in 8.3.4.

In Table 6.1, we show the recovery ratio (RE) and false positive ratio (FP) of faces in our reconstructed mesh. Note that we could recover over 99% of faces in the original mesh, while only having under 1% of false faces. Please see

Table 6.1: Mesh reconstruction results.

-	Bunny	Dragon	Buddha
RE	99.78%	99.72%	99.64%
FP	0.00%	0.55%	0.84%

Appendix E.5.1 for more details. This result successfully validates our differentiable formulation, but also reveals its limitation in reconstructing some abnormal triangles in the original mesh, such as long, thin triangles.

6.5.2 Point Cloud & Multi-View Reconstruction

In this experiment, we aim to reconstruct a mesh from partial geometric data, such as (oriented) point clouds or multi-view images. For point cloud reconstruction, we sampled 100K points from the ground truth mesh. We can additionally use point orientations, if they are available. For multi-view reconstruction, we rendered diffuse and depth images of the ground truth mesh from 64 view points. In Appendix E.5, we illustrated the example inputs for these experiments. Also, please see Appendix E.4 to see the initialization and densification strategy we took in these experiments.

To validate our approach, we compare our results with various approaches. When it comes to point cloud reconstruction, we first compare our result with classical Screened Poisson Surface Reconstruction (PSR) method [16]⁵. Then, to compare our method with optimization based approach, we use recent VoronoiMesh [17] method. Note that these two methods are not tailored for open surfaces. To compare our method also for the open surfaces, we use Neural Dual Contouring (NDC) [19], even though it is learning-based approach. Finally, for multi-view reconstruction task, we compare our

⁵We provide point orientations for PSR, which is optional for our method.

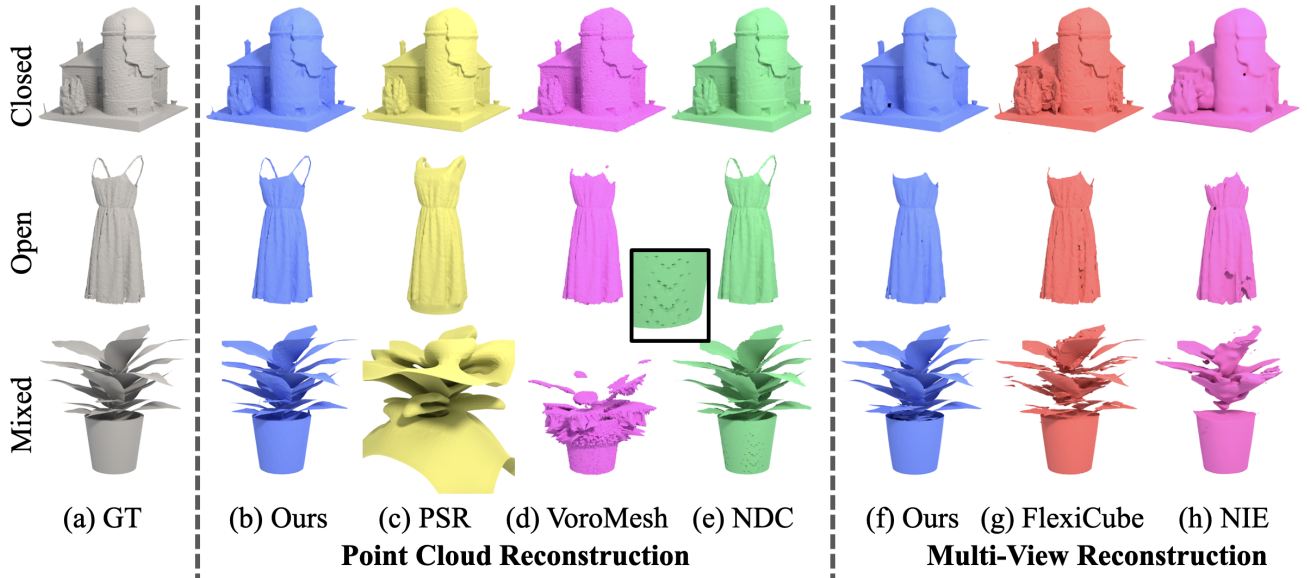


Figure 6.8: **Point cloud and multi-view reconstruction results.** (a): Ground truth mesh. (b), (f): Our method restores the original shape without losing much detail. (c), (d), (g), (h): PSR [16], VoroMesh [17], FlexiCube [11], and NIE [18] fail for open and mixed surfaces. (e): NDC [19] exhibits artifacts from grids.

results with Flexicube [11] and Neural Implicit Evolution (NIE) [18], which correspond to volumetric approaches that can directly produce meshes of varying geometric topology for given visual inputs.

In Figure 6.8, we visualize the reconstruction results along with the ground truth mesh for qualitative evaluation. For closed meshes, in general, volumetric approaches like PSR, VoroMesh, and Flexicube, capture fine details better than our methods. This is mainly because we currently have limitation in the mesh resolution that we can produce with our method. NIE, which is also based on volumetric principles, generates overly smoothed reconstruction results. However, when it comes to open or mixed mesh models, which are more ubiquitous in real applications, we can observe that these methods fail, usually with false internal structures or self-intersecting faces (Appendix E.5.2). Since NDC leverages unsigned information, it can handle these cases without much problem as ours. However, we can observe step-like visual artifacts coming from its usage of grid in the final output, which requires post-processing. Additionally, to show the versatility of our representation, we also visualize

Table 6.2: **Quantitative comparison for point cloud and multi-view reconstruction results.** Best results are written in bold.

	Methods	CD ($\times 10^{-5}$) $\downarrow$	F1 $\uparrow$	NC $\uparrow$	ECD $\downarrow$	EF1 $\uparrow$	# Verts $\downarrow$	# Faces $\downarrow$	Time (sec) $\downarrow$
PC	PSR	690	0.770	0.931	0.209	0.129	159K	319K	10.6
	VoroMesh	>1K	0.671	0.819	>1K	0.263	121K	242K	12.2
	NDC	3.611	0.874	0.936	0.022	0.421	20.7K	42.8K	3.49
	Ours (w/o normal)	3.726	0.866	0.936	0.067	0.342	3.87K	10.4K	775
	Ours (w/ normal)	3.364	0.886	0.952	0.141	0.438	3.56K	7.54K	743
MV	NIE	585	0.439	0.848	0.064	0.023	74.5K	149K	6696
	FlexiCube	273	0.591	0.881	0.039	0.152	10.9K	21.9K	56.47
	Ours	34.6	0.685	0.892	0.094	0.113	4.19K	8.80K	1434

various shapes reconstructed from oriented point clouds in Figure 6.2.

Table 6.2 presents quantitative comparisons with other methods. We used following metrics from [19] to measure reconstruction accuracy: Chamfer Distance (CD), F-Score (F1), Normal Consistency (NC), Edge Chamfer Distance (ECD), and Edge F-Score (EF1) to the ground truth mesh. Also, we report number of vertices and faces of the reconstructed mesh to compare mesh complexity, along with computational time. All values are average over 11 models that we used. In general, our method generates mesh of comparable, or better accuracy than the other methods. However, when it comes to ECD and EF1, which evaluate the edge quality of the reconstructed mesh, our results showed some weaknesses, because our method cannot prevent non-manifold edges yet. However, our method showed superior results in terms of mesh complexity – this is partially due to the use of weight regularization. Please see Appendix E.5.3 to see how the regularization works through ablation studies. Likewise, our method shows promising results in producing compact and accurate mesh. However, we also note that our method requires more computational cost than the other methods in the current implementation.

Before moving on, we present an experimental result about shape interpolation using DMesh in Figure 6.9. We used multi-view images to reconstruct a torus first, and then optimized the DMesh

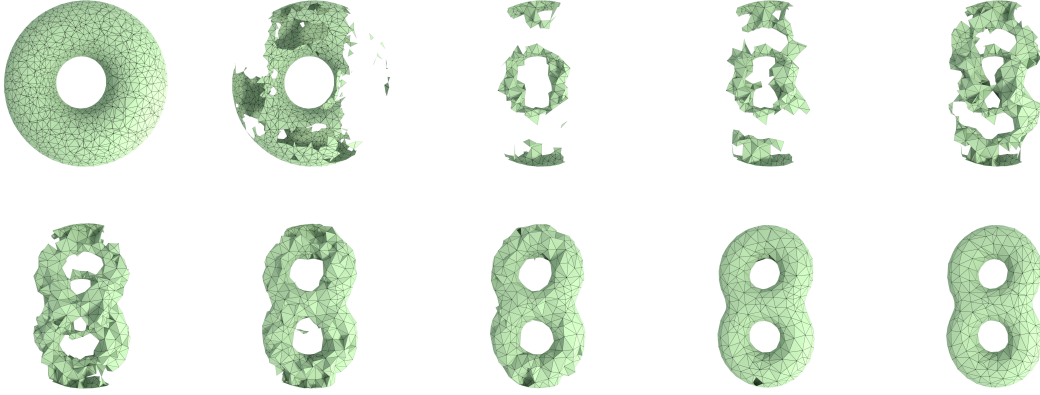


Figure 6.9: ($\rightarrow$) **Shape interpolation using DMesh exhibiting topology change.** After fitting DMesh to a torus (upper left), we optimize it again to reconstruct a double torus (lower right), which has a different genus. We use multi-view images for the optimization.

again to fit a double torus. The results show that DMesh effectively reconstructs the double torus, even when initialized from a converged single torus, highlighting the method’s robustness to local minima. However, this also indicates that our representation lacks meaningful shape interpolation, as it does not assume any specific shape topology.

6.6 Conclusion

6.6.1 Limitations

As shown above, our method achieves a more effective and complete forms of differentiable meshes of various topology than existing methods, but still has several limitations to overcome.

Computational Cost. Currently, the resolution of DMesh is limited by computational cost. Although our theoretical relaxation and CUDA implementation reduce this burden, processing meshes with over 100K vertices remains challenging due to the computational bottleneck of constructing the WDT at each optimization step. In Figure 6.10, we analyze computational costs relative to the number

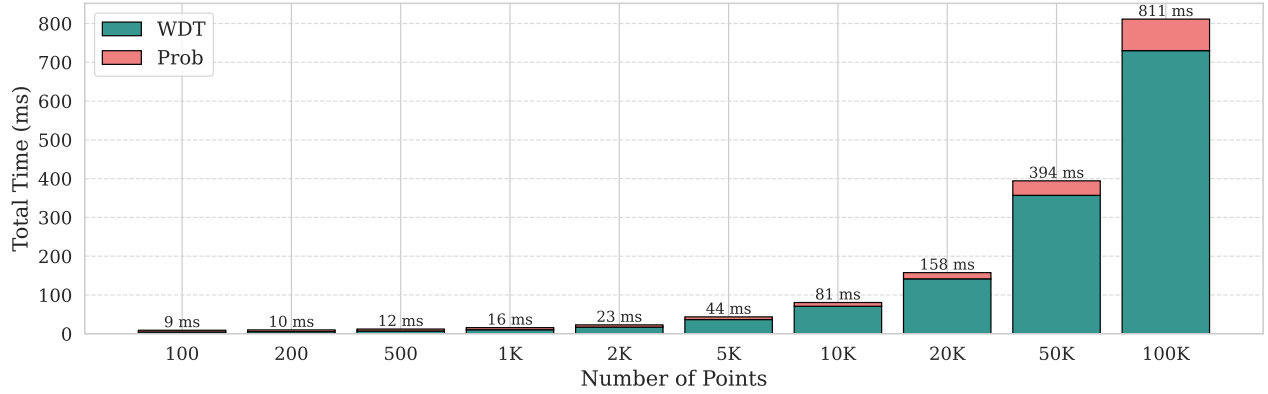


Figure 6.10: **Analysis of computational cost for computing face existence probabilities** ($\Lambda(F)$). The computational cost rises sharply beyond 20K points, with most of the time spent on WDT construction (“WDT”), while the probability computation (“Prob”) requires significantly less time.

of points. As shown, costs rise sharply beyond 20K points, with WDT construction consuming most of the time. This limits our method’s ability to handle high resolution mesh.

Non-Manifoldness. As we have claimed so far, DMesh shows much better generalization than the other methods as it does not have any constraints on the shape topology and mesh connectivity. However, due to this relaxation of constraint, we can observe spurious non-manifold errors in the mesh, even though we adopted measures to minimize them (Appendix E.4.3).

Specifically, an edge must have at most two adjacent faces to be a "manifold" edge. Similarly, a "manifold" vertex should be adjacent to a set of faces that form a closed or open fan. We refer to edges or vertices that do not satisfy these definitions as "non-manifold." In our results, we found that 5.50% of edges and 0.38% of vertices were non-manifold for point cloud reconstruction. For multi-view reconstruction, 6.62% of edges and 0.25% of vertices were non-manifold. Therefore, we conclude that non-manifold edges are more prevalent than non-manifold vertices in our approach.

6.6.2 Future Work

To address the computational cost issue, we can explore methods that reduce reliance on the WDT algorithm, as its cost increases significantly with the number of points. This is crucial since representing complex shapes with fine details often requires over 100K vertices. To tackle the non-manifold issue, we could integrate approaches based on (un)signed distance fields [11, 12] into our method, ensuring manifold mesh generation. Finally, future research could extend this work to solve other challenging problems, such as 3D reconstruction from real-world images, or applications like generative models for 3D shapes. This could involve encoding color or texture information within our framework, opening up exciting new directions for exploration.

These limitations highlight a fundamental trade-off in differentiable mesh representations between expressiveness, computational efficiency, and geometric regularity. While **DMesh** establishes a flexible and general framework for gradient-based reasoning over mesh connectivity and topology, its current formulation leaves room for improvement in scalability and robustness. Motivated by these observations, the next chapter introduces **DMesh++**, which refines the core ideas of **DMesh** to achieve more efficient and scalable differentiable mesh optimization.

Chapter 7: DMesh++: An Efficient Differentiable Mesh for Complex Shapes

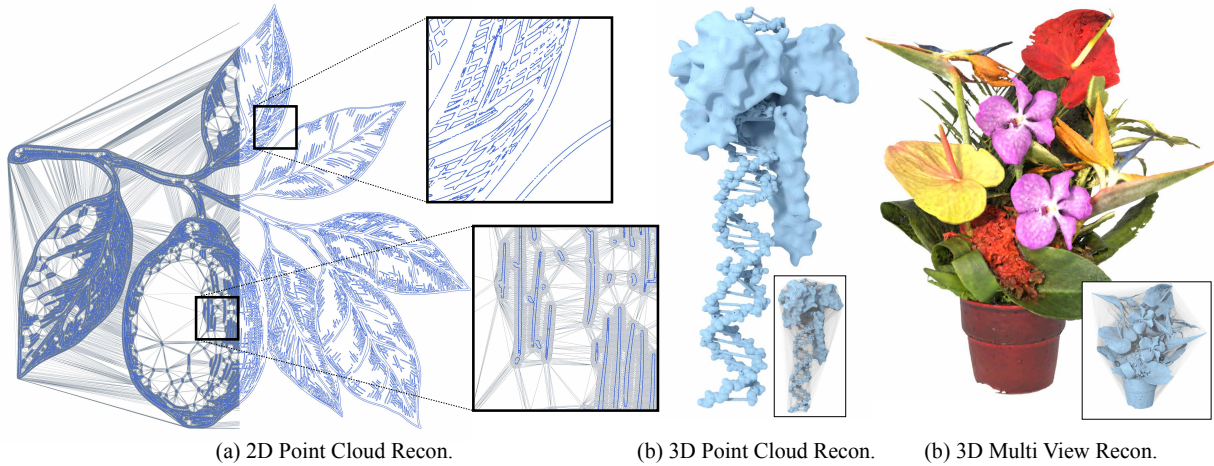


Figure 7.1: **DMesh++ for complex 2D and 3D shapes.** DMesh++ encodes all geometric and topological information into continuous point features. (a) By optimizing these point features, DMesh++ is able to reconstruct complex 2D drawings from sample points. (b) This approach is also applicable to 3D, where it reconstructs the complex geometric structure of DNA from a point cloud. (c) By incorporating additional color features, DMesh++ can reconstruct complex, colored 3D shapes from multi-view images. For each result, the “*imaginary*” part is rendered in gray, while the “*real*” part—which defines the final mesh—is rendered in other colors. ([section 7.3.1](#)).

While DMesh establishes a flexible differentiable representation for mesh connectivity and topology, its computational cost limits scalability to high-resolution shapes. This chapter addresses this limitation by introducing an efficient differentiable mesh processing framework that preserves the expressive power of probabilistic mesh connectivity while significantly improving computational and memory efficiency. Building on the foundations of DMesh, the proposed method enables practical gradient-based optimization for complex geometric structures at scale.

7.1 Introduction

Among various possible shape representations, a mesh is often favored for a wide range of downstream tasks due to its efficiency, versatility, and controllability. A mesh is defined by its vertices' position and their connectivity in the form of edges and faces. This connectivity is discrete in nature, and also the number of possible connectivities grows exponentially with the number of points, which prevents meshes from being differentiable shape representations (fig. 7.2). To address this, recent data-driven efforts have attempted to predict mesh connectivity using Transformer-based models [273–276]. However, these methods face inherent challenges with robustness to outlier meshes, potential self-intersections, and high computational costs.

On another route, Son et al. [20] introduced a new form of differentiable mesh called DMesh, which is essentially a probabilistic approach. For a given set of points, they explicitly compute probabilities for possible face combinations to exist on the mesh based on the continuous point-wise features. This approach minimizes several mesh degeneracies, and is free from outliers, as it is not data-driven (fig. 7.2). Therefore, this probabilistic approach opens up a new venue to adopt meshes in a machine learning pipeline, such as generative models [277, 278]. However, it suffers from excessive computational cost when the number of points increases (fig. 7.7), which limits its applicability for representing complex shapes with detailed structures.

In this work, we introduce DMesh++, which overcomes the computational limitations of DMesh while retaining its core advantages. To that end, we present *Minimum-Ball* algorithm. While the computational cost to evaluate face probability is $O(N)$ for DMesh, where N is the number of points that define the mesh, our *Minimum-Ball* algorithm has $O(\log N)$ computational cost (section 7.3.2 and fig. 7.7).

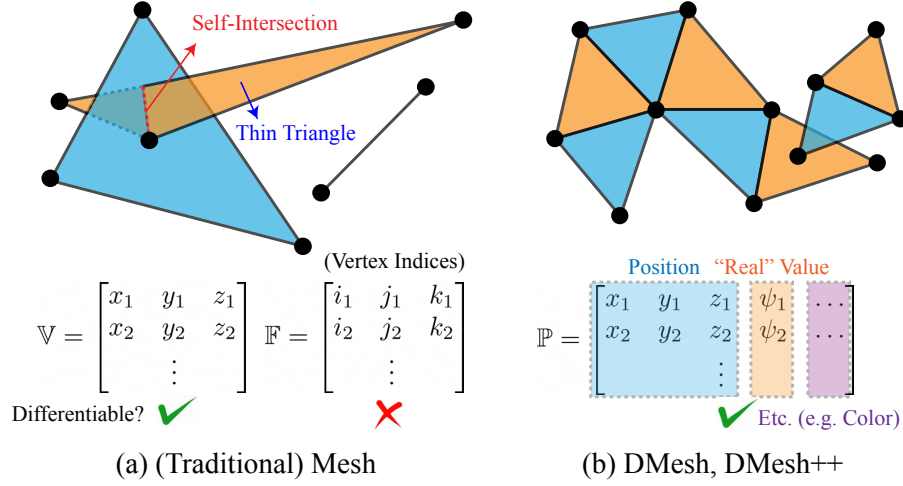


Figure 7.2: **Conceptual comparison of traditional mesh and variants of DMesh [20].** Traditional meshes employ a non-differentiable, discrete data structure, $\mathbb{F}$, to store vertex indices that define connectivity, whereas DMesh++ encodes connectivity and additional information into continuous point-wise features, $\mathbb{P}$. Mesh generated from DMesh++ avoids several degeneracies—such as self-intersections and thin triangles—that can compromise its suitability for downstream applications.

The direct application of DMesh++ is a reconstruction task. It effectively reconstructs complex 2D and 3D meshes of diverse topology from point clouds or multi-view images (figs. 7.9, 7.10 and 8.1). During the optimization of continuous point-wise features, we observe dynamic topological changes in the mesh that recover the target shape (fig. 7.3).

To summarize, our contributions are the following:

- We present DMesh++, an enhanced version of DMesh [20], which overcomes its computational bottlenecks by employing the *Minimum-Ball* algorithm.
- We propose a reconstruction algorithm that incorporates efficient loss formulations and additional mesh operations to effectively recover 2D and 3D shapes from point clouds or multi-view images.
- We validate our approach on 500+ mesh models with diverse topology, which are collected from

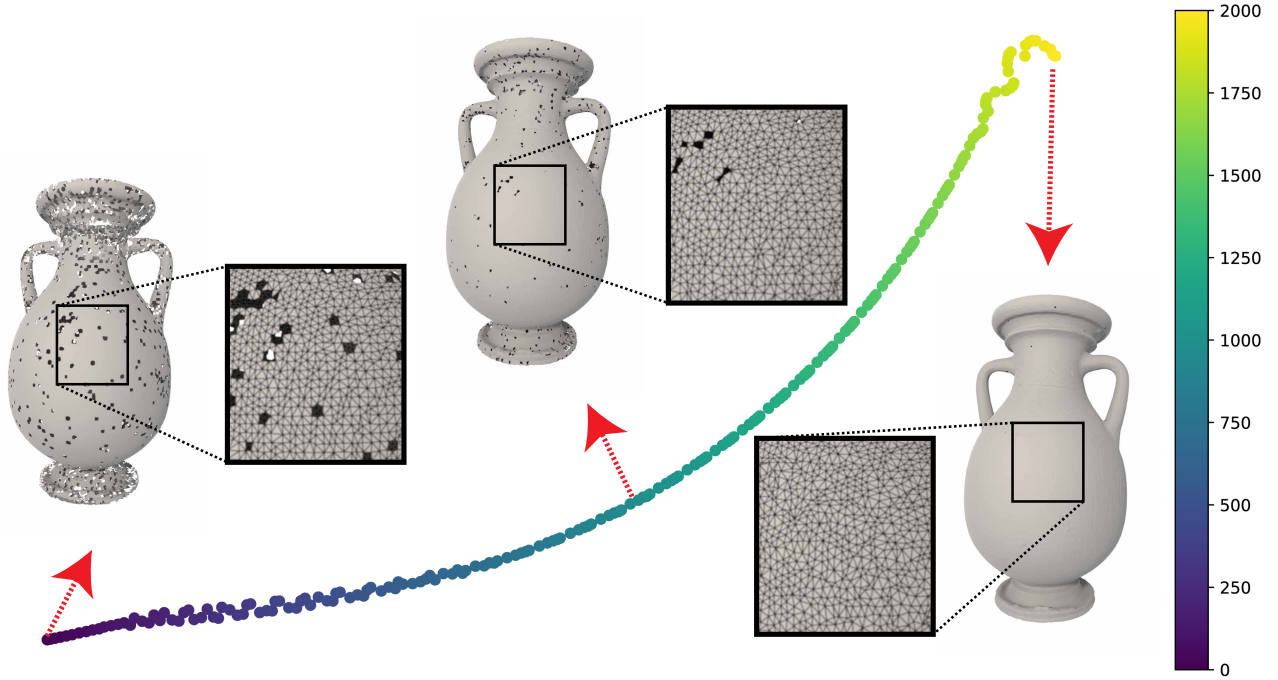


Figure 7.3: **Mesh topology change during 3D point cloud reconstruction of a vase.** We visualize the evolution of point-wise features by projecting them onto a 2-dimensional space using T-SNE [21] at each optimization step (ranging from 0 to 2K). As these features continuously evolve, the mesh undergoes discrete topological changes to progressively recover the target shape.

Thingi10K [1] and Objaverse [2] dataset.

7.2 Related Work

While meshes offer an efficient and flexible representation of shapes, they are mainly constrained by their connectivity issues, which limit their applicability in machine learning. To address these challenges, shape inference in machine learning has evolved through three stages.

Using Alternative Differentiable Shape Representations. Rather than handling mesh directly, some prior work extract mesh from alternative differentiable shape representations. Neural implicit representations, like (un)signed distance fields [5–8, 133–137], encode distance fields in neural networks, and

use iso-surface extraction algorithms [22,138,279] to generate the final mesh. Another method encodes distance directly into spatial points and applies differentiable iso-surface extraction [9–12,18,136,262]. While often more efficient, these methods typically cannot handle open surfaces; though [12] does, it cannot represent non-orientable geometries. Gaussian Splatting [280] also encodes visual data as spatial “splats” but lacks the geometric accuracy of implicit functions.

Inferring Meshes Differentiably. The main challenge in differentiable mesh handling is the exponential growth of possible vertex connections as vertex count increases. To simplify this challenge, most prior works assume fixed registration and permit only local connectivity changes [3,4,24,159–161]. Recently, data-driven approaches have aimed to overcome these limitations by training generative models [273–276] that predict vertex connectivity from point clouds. Specifically, SpaceMesh [276] ensures combinatorial manifold mesh generation. However, these models struggle with outliers and self-intersections.

Designing A Differentiable Form of Mesh. Son et al. [20] recently introduced DMesh, a differentiable mesh formulation using a probabilistic approach. DMesh augments each point with two continuous values, along with its position (fig. 7.2), and applies a “tessellation” function in eq. (7.1) to deterministically generate a mesh from a point set. This method adapts to various geometric topologies, including non-orientable open surfaces, and avoids self-intersections. With DMesh, optimizing or inferring only point-wise features is sufficient to generate the mesh.

However, DMesh’s tessellation function is slow due to its reliance on Weighted Delaunay Triangulation (WDT), which has a practical time complexity of $O(N)$ for N points using the CGAL package [272]. For $N = 100K$ in 3D, the runtime can reach up to 800 milliseconds (fig. 7.7), limiting DMesh’s applicability for complex shapes requiring finer detail. Moreover, it is hardly possible to

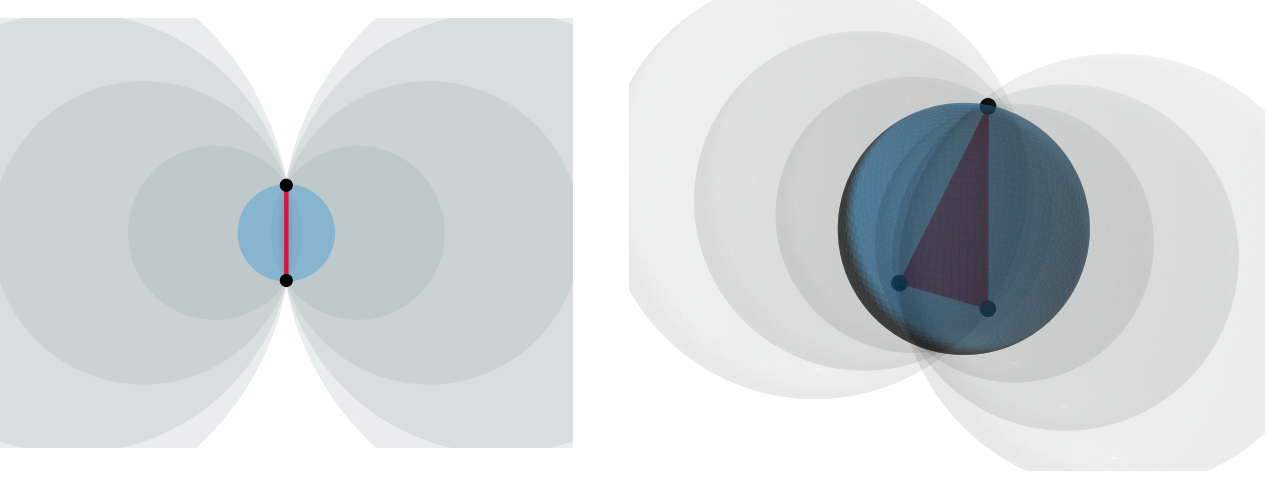


Figure 7.4: **Bounding balls of a face F (red) in 2D (left) and 3D (right).** The minimum bounding ball (B_F) is rendered in blue, while the others are rendered in gray.

accelerate WDT using parallelization, because of its inherent race conditions. Therefore, in this work, we eliminate WDT, and propose a more efficient differentiable mesh formulation.

7.3 Formulation

In this section, we first provide the high-level formulation for computing probability of a face to exist in the mesh. Then, we introduce *Minimum-Ball* (section 7.3.2), which is our primary algorithm.

7.3.1 Preliminary

In this work, we refer to a $(d-1)$ -simplex in d -dimensional space as a “face” (e.g., a line segment for $d = 2$ or a triangle for $d = 3$). DMesh [20] tessellates d -dimensional convex space using faces, with the actual surface on “real” faces and “imaginary” faces enclosing the “real” part to support the convex space (fig. 8.1).

In DMesh, each point is a $(d+2)$ -dimensional vector: the first d values denote position, while the remaining two represent the Weighted Delaunay Triangulation (WDT) weight (w) [281] and real

value (ψ) . The $\psi \in [0, 1]$ of a point indicates whether it lies on the shape, specifically if $\Psi(p) > 0.5$, where $\Psi(p)$ is ψ of a point p .

For a point set $\mathbb{P}$, let $\mathbb{F}_{wdt}$ represent the faces in WDT of $\mathbb{P}$. DMesh then introduces a “**tessellation**” function to determine if a face F exists on the mesh:

$$T_{DMesh}(\mathbb{P}, F) = (F \in \mathbb{F}_{wdt}) \wedge (\min_{p \in F} \Psi(p) > 0.5). \quad (7.1)$$

DMesh++ introduces an alternative tessellation function for faster processing. By removing the need for WDT, we eliminate the WDT weight and represent each point as a $(d+1)$ -dimensional vector, $(x_1, \dots, x_d, \psi)$ ¹. In place of WDT, we implement a faster scheme called the *Minimum-Ball* condition (Definition 7.3.1) for defining the tessellation function. Letting $\mathbb{F}_{min}$ represent the set of faces that meet this condition, we define the tessellation function as

$$T_{DMesh++}(\mathbb{P}, F) = (F \in \mathbb{F}_{min}) \wedge (\min_{p \in F} \Psi(p) > 0.5). \quad (7.2)$$

In our differentiable framework, we compute probability of F to satisfy these two conditions: Λ_{min} and Λ_{real} , respectively. Then, we compute the final probability of F to exist on the mesh as $\Lambda(F) = \Lambda_{min}(F) \times \Lambda_{real}(F)$. For Λ_{real} , we use differentiable min operator as DMesh. In the next section, we explain how we define Λ_{min} .

¹Points could carry additional features, such as color (fig. 7.2).

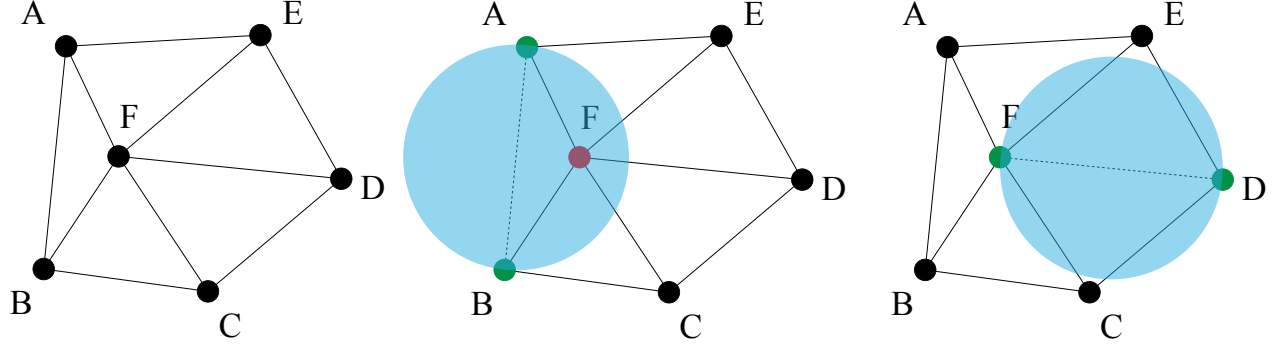


Figure 7.5: **Minimum-Ball condition in 2D.** In the left, 2D Delaunay Triangulation (DT) of 6 points is given. In middle and right figure, we render B_F for two faces $(\overline{AB}, \overline{DF})$ in blue.

7.3.2 Minimum-Ball Algorithm

The mesh generated by DMesh’s tessellation function in [eq. \(7.1\)](#) is free from self-intersections because $\mathbb{F}_{wdt}$ itself is free from them. Additionally, it minimizes the occurrence of thin triangles—undesirable in many downstream tasks such as physics simulations—thanks to the properties of the WDT. However, computing the tessellation function is computationally expensive, as it requires calculating the WDT to define $\mathbb{F}_{wdt}$. In designing our *Minimum-Ball*, we aimed to eliminate this computational bottleneck while preserving the favorable properties related to self-intersection avoidance and triangle quality. In the following, we demonstrate that *Minimum-Ball* satisfies these requirements.

For a given set of points $\mathbb{P} \in \mathbb{R}^d$ and a face $F = \{p_1, p_2, \dots, p_d\} \subset \mathbb{P}$, we define a bounding ball of F as a d -dimensional ball that goes through every point of F . Note that this bounding ball is not unique, but there exists a unique minimum bounding ball, which has the minimum radius among every bounding ball. We name it as B_F ([fig. 7.4](#)). Then, we define $\mathbb{F}_{min}$ as a set of faces whose minimum bounding ball does not contain any other point in $\mathbb{P}$.

Definition 7.3.1 $F \in \mathbb{F}_{min}$ if and only if there is no point in $\mathbb{P}$ that lies (strictly) inside B_F .

Note that we can ignore points in F , as they are located on the boundary of B_F . In [fig. 7.5](#), we render a 2D case, where $\overline{AB}$ does not satisfy this definition because of F . In contrast, $\overline{DF}$ satisfies this condition. Then, we can observe that $\mathbb{F}_{min}$ is a subset of faces in Delaunay Triangulation (DT) of $\mathbb{P}$ ($\mathbb{F}_{dt}$).

Lemma 7.3.2 $F \in \mathbb{F}_{min} \Rightarrow F \in \mathbb{F}_{dt}$.

Proof. By definition, a face F is in $\mathbb{F}_{dt}$ if there is a bounding ball of F that does not contain any other point in $\mathbb{P}$ [\[282\]](#). If the face F is in $\mathbb{F}_{min}$, its minimum bounding ball satisfies this condition. Thus F is in $\mathbb{F}_{dt}$.

Note that $\mathbb{F}_{dt}$ is also free from self-intersections as $\mathbb{F}_{wdt}$, and thus is $\mathbb{F}_{min}$. Furthermore, it inherently minimizes the number of thin triangles, as guaranteed by DT. However, note that $\mathbb{F}_{min}$ does not necessarily tessellate the entire convex shape, as there could be faces in $\mathbb{F}_{dt}$ that are not in $\mathbb{F}_{min}$ (e.g., $\overline{AB}$ in [fig. 7.5](#)).

Now, based on [Definition 7.3.1](#), we can check if F is in $\mathbb{F}_{min}$. Let us denote the center and radius of B_F as $B_F^c \in \mathbb{R}^d$ and B_F^r . We can compute these values in a differentiable way ([Appendix F.1.2](#)). Then, we can compute the signed distance between B_F and $\mathbb{P}$ as follows:

$$d(B_F, \mathbb{P}) = \min_{p \in \mathbb{P} - F} ||p - B_F^c|| - B_F^r. \quad (7.3)$$

As shown above, we can easily find $d(B_F, \mathbb{P})$ by finding the nearest point of B_F^c in $\mathbb{P} - F$. Using this signed distance, we can check if F is in $\mathbb{F}_{min}$ as follows.

$$F \in \mathbb{F}_{min} \Leftrightarrow d(B_F, \mathbb{P}) > 0. \quad (7.4)$$

Then, we define Λ_{min} with sigmoid function as

$$\Lambda_{min}(F) = \sigma(d(B_F, \mathbb{P}) \cdot \alpha_{min}), \quad (7.5)$$

where α_{min} is a constant (Appendix F.1.3).

With this formulation, we can evaluate eq. (7.2) far more efficiently than eq. (7.1). Our method relies on a highly parallelizable nearest neighbor search algorithm², unlike the sequential WDT. While WDT has a practical time complexity of $O(|\mathbb{P}|)$, it is relatively slow. In contrast, our approach has a time complexity of $O(|F| \cdot \log |\mathbb{P}|)$, where $|F|$ is the number of query faces to evaluate. However, by parallelizing the nearest neighbor search across query faces, especially on GPU, this complexity effectively reduces to $O(\log |\mathbb{P}|)$ ³. This allows our tessellation function to run up to 32 times faster in 3D than DMesh [20] (fig. 7.7). For optimization tasks like reconstruction, we further accelerate by periodically caching nearest neighbors for each query face (Appendix F.1.4). We provide formal algorithm in Appendix F.1.1.

7.4 Reconstruction Process

The goal of reconstruction is to optimize point-wise features so that the resulting mesh aligns with the input observation. As shown in fig. 7.3, the discrete mesh topology dynamically changes during the optimization of continuous features to better fit the given input.

fig. 7.6 provides an overview of our reconstruction process for recovering a 3D colored mesh from multi-view images. In the first stage, we initialize per-point features. If a point cloud is available,

²We used implementation of PyTorch3D [283].

³We assume that $|F|$ does not increase exponentially, which is a practical assumption as query faces are often determined by local proximity.

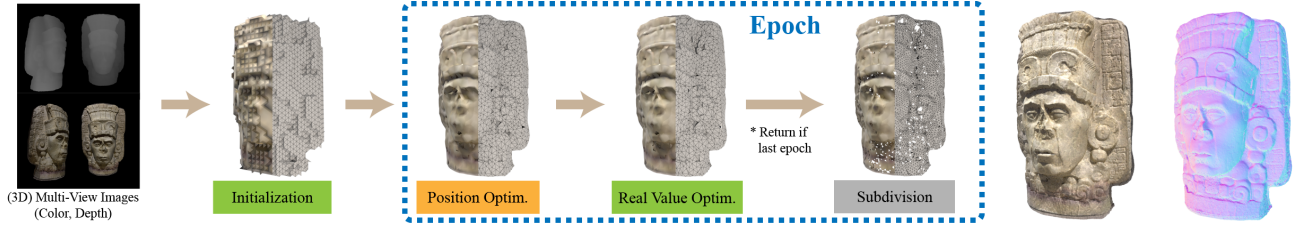


Figure 7.6: **Reconstruction process for 3D multi-view colored images of a sculpture.** In each stage, we optimize different per-point features: the **position** and the **real** (ψ), while the per-point color is refined at every stage. (Left) We display the meshes at each stage during the first epoch. (Right) We provide a rendering of the final mesh along with its view-point normal, which reveals the fine mesh details.

we use it to obtain a better starting point (see the initial mesh in [fig. 7.3](#)); otherwise, we initialize with a regular tetrahedral grid. Next, we optimize the point positions while keeping the real values fixed, followed by optimizing the real values while fixing the point positions. In these two stages, we minimize loss functions tailored to each input modality (e.g., Chamfer Distance loss for point clouds, rendering loss for images). To increase mesh complexity and capture finer details, we subdivide the mesh by inserting additional points. This process is iterated for a fixed number of epochs. Finally, to accelerate the overall process and enhance the final mesh quality, we introduce several innovations at each stage. Detailed explanations are provided in [Appendix F.2](#), due to lack of space.

7.5 Experiments

This section presents our experimental results. First, we evaluate how *Minimum-Ball* enhances the computational efficiency of the tessellation function in both 2D and 3D. Next, we demonstrate the results of a point cloud reconstruction task in 2D and 3D as a practical application. Finally, we showcase an application in 3D multi-view reconstruction. Together, these results illustrate that our method is well-suited for downstream tasks involving complex shapes. Our main algorithms are implemented in PyTorch [\[167\]](#) and CUDA [\[271\]](#). All experiments were conducted on a system with an AMD EPYC

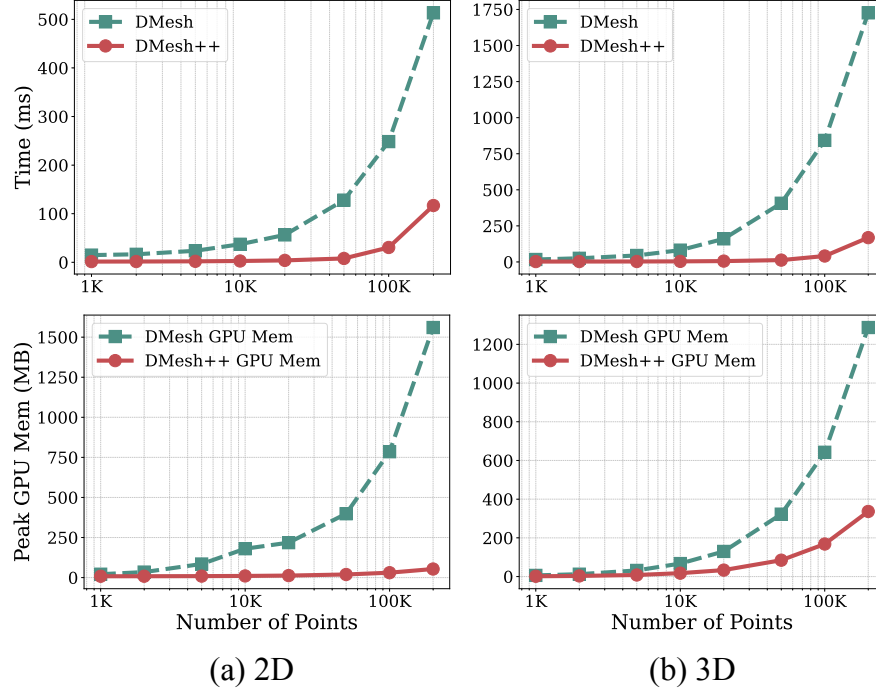


Figure 7.7: **Comparison of tessellation cost.** Our method computes face probabilities up to 16 times faster in 2D and 32 times faster in 3D than DMesh [20], while using up to 96% less GPU memory in 2D and 75% less in 3D.

7R32 CPU and an NVIDIA A10 GPU. For details, refer to Appendix F.3.

7.5.1 Tessellation Cost

We compare the computational cost of the tessellation function for DMesh [20] (eq. (7.1)) and our DMesh++ (eq. (7.2)). For both 2D and 3D scenarios, we randomly generate N points within a unit cube, find each point’s 10 nearest neighbors, and use these proximities to form potential face combinations. From these, we randomly select N faces as query faces for the tessellation function. For each value of N (ranging from 1K to 200K to reflect practical scenarios), we conducted 5 trials and averaged the computational costs.

In fig. 7.7, we compare the computational costs of DMesh and DMesh++. In terms of speed, DMesh’s performance scales linearly with the number of points in both 2D and 3D due to its sequential

WDT algorithm. In contrast, DMesh++ exhibits sub-linear scaling up to 50K points, benefiting from GPU parallelization (section 7.3.2). Beyond this range, computational costs increase more sharply because of GPU thread limitations; however, DMesh++ still processed 200K points in 117ms for 2D and 168ms for 3D. Regarding GPU memory usage, both methods scale linearly, but DMesh++ uses significantly less memory since it stores only the additional information related to the minimum balls, whereas DMesh must store all details of the power diagram on the GPU⁴.

These results demonstrate that the *Minimum-Ball* algorithm significantly enhances tessellation efficiency, enabling the effective handling of complex shapes.

7.5.2 Reconstruction Tasks

Dataset. For the 2D point cloud reconstruction task, we used vector graphics of 26 letters from four different font styles downloaded from the Google Fonts service⁵. Additionally, we employed six vector graphic images representing complex drawings from Adobe Stock⁶.

For the 3D reconstruction tasks, we randomly selected 500 models from Thingi10K [1] dataset, half of which are closed-surface models and the other half are open-surface models. Also, we manually chose 20 and 30 models from Thingi10K and Objaverse [2] dataset for better diversity. We normalized the model scale to 1 before reconstruction.

Metrics. For 2D results, we report the Chamfer Distance (CD) of the reconstructed outputs for quantitative comparison. For 3D, we assess geometric accuracy using five metrics—CD (Chamfer Distance), F1 (F1 score), NC (Normal Consistency), ECD (Edge Chamfer Distance), and EF1 (Edge

⁴Note that the 2D version of DMesh requires more memory than the 3D version, as the 2D implementation is not as optimized using CUDA.

⁵<https://fonts.google.com/>

⁶<https://stock.adobe.com/>

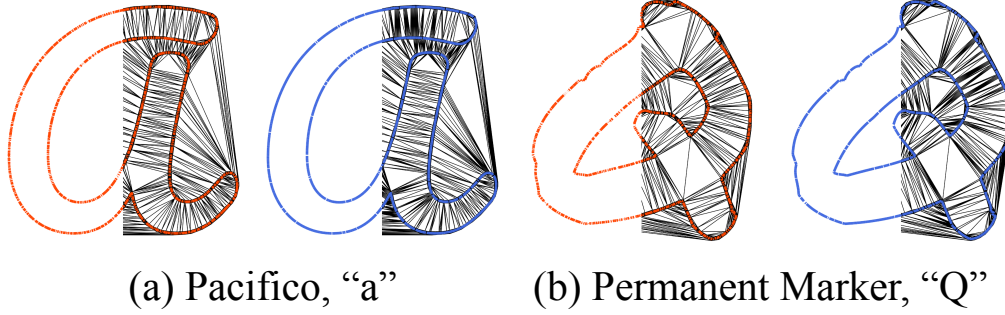


Figure 7.8: **Qualitative comparison of 2D point cloud reconstruction results.** The outputs of DMesh [20] and DMesh++ are rendered in **red** and **blue**, respectively.

Method	CD($\times 10^{-6}$) $\downarrow$	# Verts.	# Edges.	Time (sec)
DMesh [20]	1.97	2506	2245	30.39
DMesh++	1.82	2862	2793	11.33

Table 7.1: **Quantitative comparison of the 2D point cloud reconstruction task for the font dataset.** DMesh++ reconstructs 2D meshes with greater accuracy and efficiency than DMesh.

F1 score)—following [19]. Additionally, we evaluate mesh quality using four metrics: AR (Aspect Ratio), SI (Self-Intersection ratio), NME (Non-Manifold Edge ratio), and NMV (Non-Manifold Vertex ratio). Finally, we provide statistics on mesh complexity (e.g., vertex and face counts) along with the reconstruction time.

7.5.2.1 2D Point Cloud Reconstruction

In this task, we aim to reconstruct a 2D mesh from a 2D point cloud to demonstrate that the DMesh++ formulation is applicable in 2D. We primarily compare our reconstruction results on a font dataset with those obtained by DMesh [20], whose formulation is also easily extendable to 2D. As input, we sampled 1,000 points from each spline curve composing the font and downsampled the entire point cloud using a grid with a cell size of 0.005.

In [table 7.1](#), we present a quantitative comparison of the reconstruction results. We observe that

DMesh++ reconstructs 2D meshes more faithfully than DMesh in terms of CD loss, while also running $2.6\times$ faster. In [fig. 7.8](#), we render the reconstructed 2D meshes for two examples. The results show that DMesh++ produces significantly fewer holes compared to DMesh, which is consistent with the CD loss comparison. However, some of these holes are inevitable, as we mainly reconstruct our shape with CD loss and there are places that lack points.

Additionally, we reconstructed several complex 2D drawings from point clouds to demonstrate the computational efficiency of DMesh++. Since these drawings contain finer details than the fonts, we downsampled the entire point cloud using a grid with a cell size of 0.001. When we attempted to reconstruct these drawings with DMesh, the GPU memory consumption became prohibitively high, resulting in an error during reconstruction. Therefore, for qualitative evaluation, we report only the results of DMesh++ in [fig. 8.1](#) and Appendix [F.3.2.2](#).

Before moving on, we’d like to introduce an experimental algorithm that is applicable to 2D mesh optimization called the *Reinforce-Ball* algorithm, which could be used for producing an efficient mesh that adapts to local geometry. Please see Appendix [F.4](#) for more details.

7.5.2.2 3D Point Cloud Reconstruction

In this task, we reconstruct a 3D mesh from a dense 3D point cloud. As input, we sampled 200K points from the ground truth mesh using the Poisson disk sampling algorithm [\[284\]](#) implemented in MeshLab [\[25\]](#). For comparison, we employed other widely used optimization-based point cloud reconstruction methods, including Screened Poisson Surface Reconstruction (PSR) [\[16\]](#), VoroMesh [\[17\]](#), PoNQ [\[26\]](#), and DMesh [\[20\]](#). For PSR, DMesh, and PoNQ, we used oriented point clouds for reconstruction, while for DMesh++ we tested both unoriented and oriented point clouds. We used the

Method	Geometric Accuracy					Mesh Quality				Statistics		
	CD($\times 10^{-3}$) $\downarrow$	F1 $\uparrow$	NC $\uparrow$	ECD $\downarrow$	EF1 $\uparrow$	AR $\downarrow$	SI $\downarrow$	NME $\downarrow$	NMV $\downarrow$	# Verts.	# Faces.	Time (sec)
VoroMesh [17]	19.591	0.352	0.855	0.054	0.072	145.7	0	0.001	0	64561	129338	11
DMesh++	0.034	0.471	0.919	0.063	0.094	1.765	0	0.130	0.003	25415	58537	282
PSR [16]	10.164	0.392	0.943	0.302	0.026	5.218	0	0	0	139857	279739	4
PoNQ [26]	1.578	0.402	0.934	0.056	0.090	2.288	0	0.002	0	47254	94664	32
DMesh [20]	0.154	0.289	0.921	0.077	0.069	1.961	0	0.103	0.002	5815	13088	1147
DMesh++	0.033	0.480	0.938	0.060	0.116	1.814	0	0.087	0.004	25396	55546	282

Table 7.2: **Quantitative comparison of 3D point cloud reconstruction results over 50 manually chosen models from Thingi10K [1] and Objaverse [2].** We highlight the **best** results for each metric. The upper panel compares methods that use *unoriented* point clouds (VoroMesh, DMesh++), while the lower panel displays methods that use *oriented* point clouds (PSR, PoNQ, DMesh, DMesh++).

Method	CD($\times 10^{-4}$) $\downarrow$	F1 $\uparrow$	NC $\uparrow$	ECD $\downarrow$	EF1 $\uparrow$
PSR [16]	12.1 / 16.3	0.47 / 0.45	0.97 / 0.96	0.45 / 0.37	0.01 / 0.01
PoNQ [26]	2.44 / 7.86	0.48 / 0.45	0.97 / 0.95	0.01 / 0.04	0.26 / 0.22
DMesh [20]	2.82 / 3.29	0.25 / 0.22	0.94 / 0.93	0.01 / 0.01	0.15 / 0.12
DMesh++	0.37 / 0.37	0.48 / 0.47	0.96 / 0.95	0.02 / 0.02	0.25 / 0.23

Table 7.3: **Quantitative comparison of 3D point cloud reconstruction results over 500 randomly chosen models from Thingi10K [1].** The results over closed surfaces and open surfaces are shown together: (closed / open). We highlight the best results for two different categories with **red** and **blue**, respectively.

PSR implementation available in MeshLab. For DMesh, we used its default settings; for VoroMesh and PoNQ, we employed a grid size of 128 and optimized for 1000 epochs to achieve the best results.

We report the quantitative comparisons in tables 7.2 and 7.3. In table 7.2, results are averaged over the 50 manually chosen models and split into two categories: methods using *unoriented* versus *oriented* point clouds. In both categories, DMesh++ performs best or is comparable to other methods in geometric accuracy and triangle aspect ratio. Compared to DMesh, DMesh++ outperforms all metrics while handling $4.2\times$ more faces with a 76% reduction in computation time. Furthermore, DMesh++ achieves significantly better CD, F1, and EF1 scores than PSR, VoroMesh, and PoNQ, which struggle with open surfaces. Separate evaluations in table 7.3 over 500 randomly chosen models confirm that

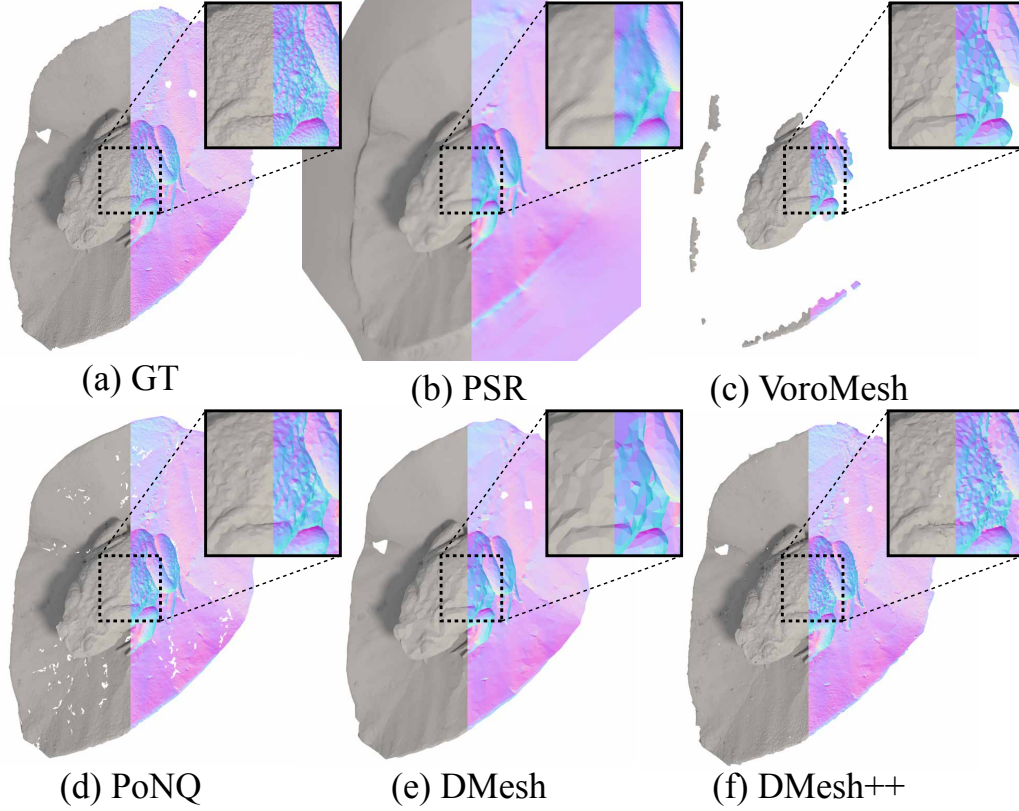


Figure 7.9: **Qualitative comparison of 3D point cloud reconstruction results for a toad sitting on a leaf.** For each result, we render its diffuse image on the left, and view-point normals on the right.

DMesh++ largely outperforms on open surfaces and performs comparably or better on closed surfaces.

These quantitative results align with the qualitative comparison in [fig. 7.9](#), where a toad sitting on a thin leaf is reconstructed . While PSR and VoroMesh fail to reconstruct the open surface of the leaf and PoNQ produces many holes, DMesh captures the overall geometry but misses fine details. DMesh++ successfully recovers both the overall shape and the intricate details. For qualitative comparison on a closed surface, see [fig. F.7](#).

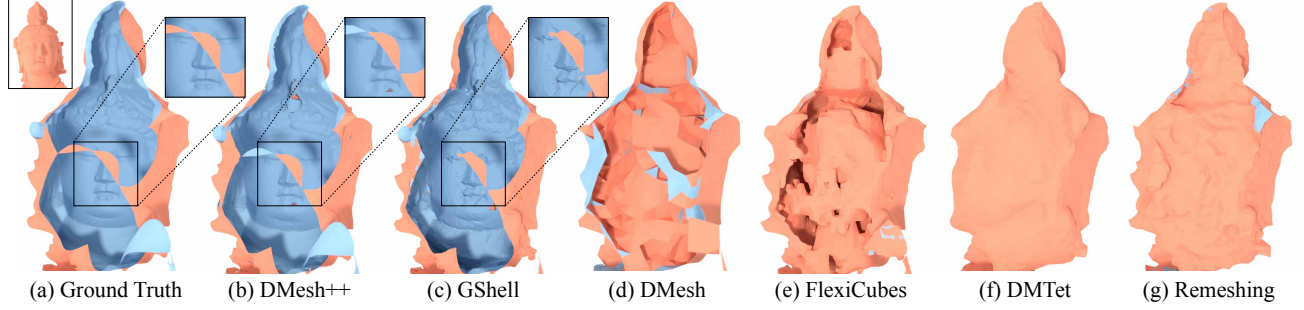


Figure 7.10: **Qualitative comparison of 3D multi-view reconstruction results for open surface.** Here we illustrate from back of an open surface model (the front view is rendered at the left top of (a)). Colors represent **inside** and **outside facing surfaces**.

Method	Geometric Accuracy					Mesh Quality				Statistics		
	CD($\times 10^{-3}$) $\downarrow$	F1 $\uparrow$	NC $\uparrow$	ECD $\downarrow$	EF1 $\uparrow$	AR $\downarrow$	SI $\downarrow$	NME $\downarrow$	NMV $\downarrow$	# Verts.	# Faces.	Time (sec)
Remeshing [3]	0.977	0.343	0.907	0.088	0.044	1.562	0.220	0	0	13273	26540	25
DMTet [9]	1.395	0.191	0.868	0.145	0.032	6.175	0	0	0	20549	41131	201
FlexiCubes [11]	3.493	0.290	0.880	0.091	0.038	2.093	0.011	0	0.005	14811	28882	88
GShell [12]	5.807	0.326	0.904	0.116	0.047	2.793	0.047	0	0.004	14587	28054	209
DMesh [20]	0.697	0.328	0.898	0.104	0.045	1.820	0	0.042	0.002	2461	5058	772
DMesh++	0.342	0.360	0.923	0.074	0.059	1.639	0	0.025	0.036	12507	23727	205

Table 7.4: **Quantitative comparison of 3D multi-view reconstruction results over 50 manually chosen models from Thingi10K [1] and Objaverse [2].** We highlight the **best** results and the **second best** results for each evaluation metric.

7.5.2.3 3D Multi-View Reconstruction

In this task, we reconstruct a mesh from multi-view images of a target object, assuming full knowledge of the rendering model and lighting conditions. Specifically, we employ simple Phong shading [285] with a directional light from the camera to the object for rendering the ground truth images. We generate (512×512) diffuse and depth maps of the object from 64 viewpoints (fig. F.8) to supervise the reconstruction process. Note that we omit colors and textures in this experiment to focus solely on geometric quality.

For comparison, we evaluated five mesh reconstruction algorithms: Remeshing [267], DMTet [9], FlexiCubes [11], GShell [12], and DMesh [20]. We optimized each method to produce the best-quality

Method	CD($\times 10^{-4}$) $\downarrow$	F1 $\uparrow$	NC $\uparrow$	ECD $\downarrow$	EF1 $\uparrow$
REM [3]	58.0 / 27.5	0.33 / 0.32	0.90 / 0.90	0.06 / 0.07	0.11 / 0.09
DMT [9]	36.2 / 65.1	0.21 / 0.21	0.89 / 0.88	0.17 / 0.13	0.03 / 0.03
FLE [11]	14.5 / 34.1	0.37 / 0.35	0.92 / 0.91	0.04 / 0.06	0.10 / 0.08
GSH [12]	5.74 / 54.6	0.38 / 0.36	0.94 / 0.93	0.04 / 0.06	0.15 / 0.12
DME [20]	11.9 / 11.7	0.18 / 0.20	0.87 / 0.87	0.06 / 0.08	0.04 / 0.04
DMesh++	3.35 / 3.82	0.37 / 0.36	0.94 / 0.93	0.03 / 0.03	0.21 / 0.17

Table 7.5: **Quantitative comparison of 3D multi-view reconstruction results over 500 randomly chosen models from Thingi10K [1].** The results over closed surfaces and open surfaces are shown together: (closed / open). We highlight the best results for two different categories with **red** and **blue**, respectively.

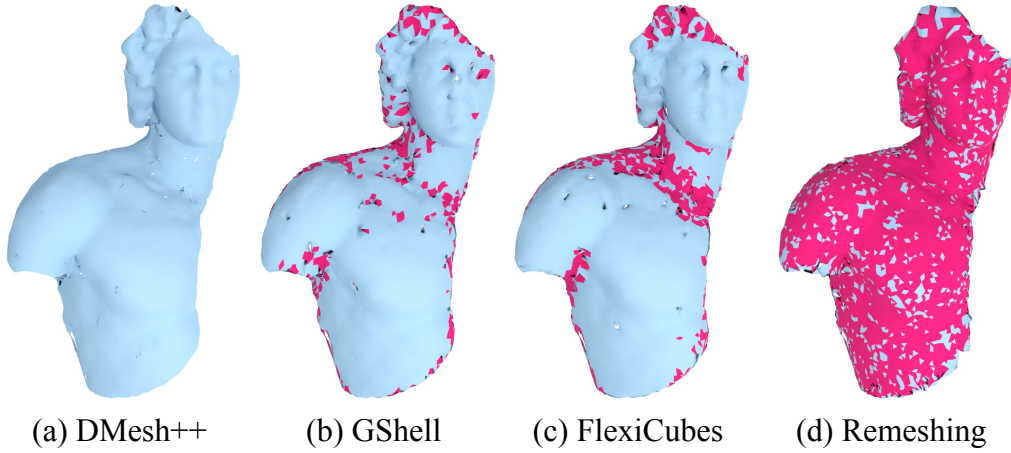


Figure 7.11: **Self-intersection of the reconstructed mesh.** The self-intersected faces of the mesh are rendered in red.

meshes with similar vertex and face counts. Specifically, we set the grid sizes to 128 for DMTet, and 80 for both FlexiCubes and GShell. For more details, please refer to Appendix F.3.4.

In tables 7.4 and 7.5 and figs. 7.10 and 7.11, we present quantitative and qualitative comparisons of the reconstruction results. In table 7.4, we observe that Remeshing, DMTet, and FlexiCubes have high CD errors, largely because they cannot represent open surfaces (fig. 7.10). This limitation also explains why Remeshing and FlexiCubes are faster than other methods. Although Remeshing achieved the best AR and avoided non-manifoldness, it generated numerous self-intersections, particularly on open surfaces (fig. 7.11).

When it comes to GShell, we found out that it usually improves CD loss by representing open surfaces through sub-surface extraction from closed templates. However, it still struggles with complex open surfaces (fig. 7.10). Also, it had a robustness issue with several outlier models – which is the main reason of high average CD, even though it achieved much better CD for most of the models. Additionally, it also suffers from self-intersections (fig. 7.11) and suboptimal AR. Compared to that, DMesh produced self-intersection free mesh with better CD and AR, as it can represent open surfaces robustly. Also, it produced much simpler mesh than the other methods. However, it sometimes produced false inner structure due to occlusion (fig. 7.10), and its largest drawback was in computational cost, limiting its utility for fine-grained reconstructions.

DMesh++ addresses this issue using the *Minimum-Ball* algorithm (section 7.3.2) and nearest neighbor caching (Appendix F.1.4). It achieves the best or comparable results across all metrics while maintaining computational costs similar to those of other methods. Furthermore, as shown in table 7.5, DMesh++ delivers superior, or at least comparable performance for both closed and open surfaces. Qualitative comparisons in figs. F.8 and 7.10 also support this observation. These results prove the robustness of DMesh++ in handling complex shapes with diverse topologies, whereas other methods are limited in one aspect or another.

Colors. As mentioned in fig. 7.2 and section 7.3.1, points can carry additional features. Here, we demonstrate that we can jointly optimize per-point colors to recover a textured shape from multi-view images. For a point on a face, the color of the point is determined by barycentric interpolation of the face vertex colors. Under the assumption that we know all the rendering and lighting conditions, we can reconstruct textured meshes from multi-view images as shown in figs. 7.6, 8.1 and 8.10. In particular, fig. 8.10 demonstrates that DMesh++ can reconstruct a small scene on which physics simulations, such as bouncing balls, can be run directly. These results highlight a promising future direction of

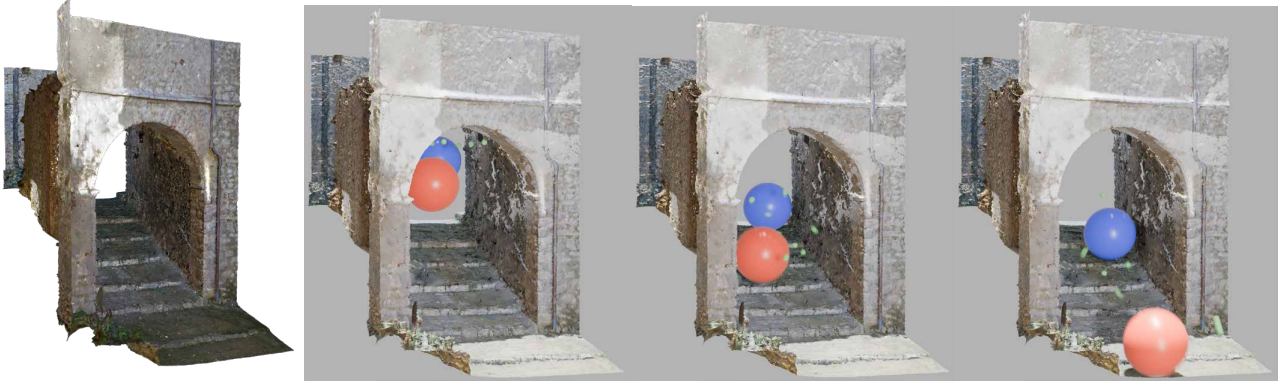


Figure 7.12: **Physics simulation on a staircase reconstructed from multi-view images.** We simulate the motion of bouncing balls directly on the mesh generated by DMesh++.

jointly optimizing additional per-point features.

7.6 Conclusion

We presented DMesh++, a probabilistic approach for efficient, differentiable mesh connectivity handling. Our *Minimum-Ball* algorithm significantly reduces computational costs, enabling DMesh++ to recover 2D and 3D shapes with diverse topologies from point clouds or multi-view images more effectively than baseline methods.

Limitations. There are areas where DMesh++ can be further improved. First, some non-manifoldness errors remain in the reconstruction results (tables 7.2 and 7.4), while observing significant improvement over DMesh [20]. We conjecture that there is a trade-off between the representation power and the occurrence of these topological errors; more careful analysis of the trade-off is needed to maintain expressiveness while eliminating errors. Second, there are application-specific challenges. Although our method yields superior 3D point cloud reconstruction results, it incurs higher computational costs (table 7.2), and it is less effective for sparse point clouds. In 3D multi-view reconstruction, we cannot use the current implementation for real-world images, as discussed in Appendix F.3.4.2.

Future Directions. For 3D point cloud reconstruction, we plan to explore alternative loss formulations to CD loss, as its computation currently dominates the overall cost.

To extend our 3D multi-view reconstruction algorithm to real-world images, we will integrate DMesh++ with alternative representations such as Gaussian Splatting (GS) [280, 286]. By appending GS features to the per-point features and optimizing them jointly, we expect to recover high-quality meshes that we can readily use for downstream tasks.

Furthermore, we envision leveraging DMesh++ to train generative models that capture complex mesh connectivity. Currently, the *Minimum-Ball* algorithm only uses point positions for connectivity. In future work, we plan to incorporate additional per-point features into this algorithm to train generative models capable of understanding diverse and intricate mesh connectivity, such as that of DNA (fig. 8.1).

We believe our work establishes an important foundation for harnessing differentiable, probabilistic mesh within the current optimization framework, and we hope it will further drive future downstream applications.

While DMesh++ enables efficient and scalable optimization over complex surface mesh connectivity, many physical and geometric reasoning tasks require explicit volumetric representations. In particular, applications such as physical simulation, collision handling, and solid modeling demand access to interior structure rather than surface geometry alone. Motivated by this need, the next chapter extends differentiable mesh optimization from surfaces to volumes by introducing a differentiable tetrahedral mesh representation.

Chapter 8: Differentiable Delaunay Tetrahedral Mesh Generation

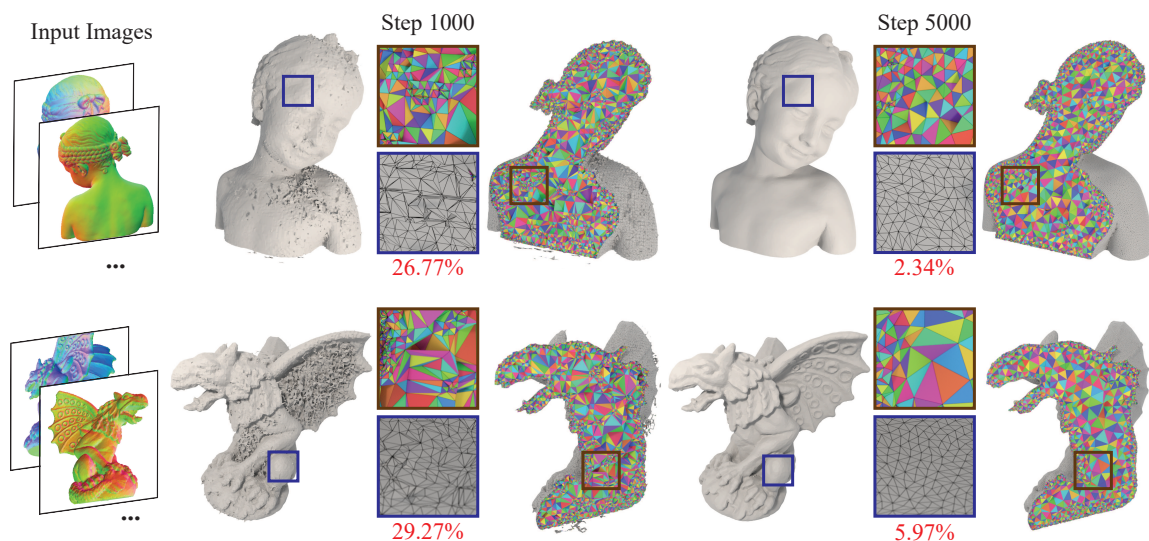


Figure 8.1: **Tetrahedral mesh reconstruction from multi-view images.** Using our differentiable tetrahedral mesh formulation, we reconstruct a high-quality volumetric mesh from multi-view images. Over the course of optimization, geometric fidelity and mesh quality improve concurrently: the recovered surface resolves fine-scale image details, and the tetrahedral elements and surface triangles become more regular. Notably, the fraction of elements whose minimum dihedral angle is below 10° (shown in red) decreases markedly. The final mesh further exhibits internal grading: fine near the boundary and coarse in the interior; thereby concentrating degrees of freedom where they matter most and yielding a more efficient representation without losing precision.

The previous chapters established efficient and scalable differentiable representations for surface meshes, enabling gradient-based reasoning over geometry and connectivity. However, many geometric processing and physical simulation tasks require explicit access to volumetric structure rather than boundary surfaces alone. This chapter addresses this need by introducing a differentiable framework for tetrahedral mesh optimization, extending gradient-informed reasoning from surface representations to volumetric domains with adaptive topological tessellation.

8.1 Introduction

3D shape modeling in machine learning has progressed rapidly, driven by advances in both *implicit* and *explicit* representations. Implicit neural fields, such as signed distance and occupancy functions, offer compact, continuous parameterizations that are naturally optimized by gradient descent and easily conditioned on images, text, or multimodal cues [127, 135, 287, 288]. In parallel, explicit representations remain prevalent due to their direct geometric access and mature differentiable operators (e.g., differentiable rendering, correspondence losses), enabling end-to-end training toward task objectives [3, 4, 20, 24, 160, 161, 289]. This diversity of representations has catalyzed impressive results in 3D reconstruction, mesh generation, and shape editing.

Despite this progress, many downstream applications in 3D graphics, VR, Robotics, and scientific computing represent geometry using *volumetric discretizations*. Finite Element Method (FEM) solvers for elasticity, heat transfer, or wave propagation require tetrahedral meshes with solver-friendly element quality (e.g., non-degenerate dihedral angles) to discretize and solve governing PDEs [290].

Related to this, a line of work has proposed differentiable pipelines that produce volumetric tetrahedral meshes [9, 11, 162], enabling direct use in FEM solvers and other downstream applications. A common limitation, however, is the reliance on a space-tiling background grid (voxel or tetrahedral) that is deformed during optimization while its *connectivity remains fixed*. Because the grid topology cannot change, and its vertices are initialized on a regular lattice, the class of representable meshes is restricted. Such fixed-topology lattices also impose practical drawbacks: they introduce many more degrees of freedom than necessary because uniform tilings must be globally fine to capture local detail. The resulting high-dimensional optimization often converges slowly and is memory-intensive. The near-uniform element distributions lead to unnecessarily large linear systems and, consequently, slow

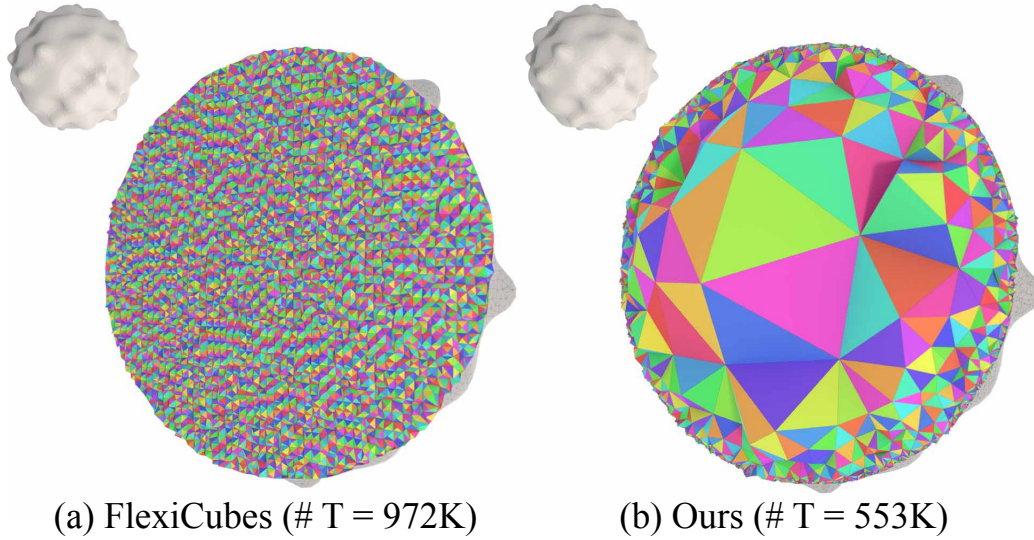


Figure 8.2: **Comparison of reconstructed tetrahedral mesh to FlexiCubes [11].** Compared with FlexiCubes, our method yields more element-efficient, internally graded tetrahedral mesh as it allows topological tessellation changes.

FEM solves. Since connectivity cannot adapt, these methods struggle to represent adaptive, anisotropic element layouts; in practice, they produce nearly uniform meshes (see the left panel of [fig. 8.2](#)).

Allowing the mesh topology to change during optimization would address these issues but is technically challenging: Re-meshing operations are discrete, and connectivity changes can introduce discontinuous gradients. Degeneracies (e.g., sliver elements) or inverted tetrahedra must be avoided to preserve physical validity. Element quality must also be maintained throughout optimization to keep downstream analysis such as FEM numerically stable.

To overcome these limitations and handle a broader class of tetrahedral meshes, we introduce a *differentiable optimization* framework grounded in Delaunay triangulation (DT). At each iteration, we re-tetrahedralize the (freely moving) vertex set with DT and then extract the volumetric mesh via a volumetric Marching Tetrahedra (MT) procedure. This design offers two key advantages: (i) DT maintains a valid connectivity as vertices move; and (ii) it promotes well-shaped elements that satisfy quality constraints to prevent degeneracy, which contributes to alleviate discontinuity issues

from connectivity changes. Consequently, from the same input modality as FlexiCubes [11], our method can produce more element-efficient, high-quality mesh, as shown in [fig. 8.2](#). Our contributions are as follows:

- We present a differentiable tetrahedral mesh optimization framework based on Delaunay triangulation (DT).
- By allowing topological grid changes and differentiability with optimization, our method represents a broader class of tetrahedral meshes than baseline approaches.
- We demonstrate efficacy by reconstructing geometrically accurate, high-quality tetrahedral meshes from signed distance fields (SDFs) and multi-view images.

8.2 Related Work

8.2.1 Tetrahedral Meshing

Tetrahedral mesh is one of the most widely used data structure in subareas of computer graphics, including volumetric rendering [146–150], fluid simulation [151–153], soft-body simulation [154–157], and fracture simulation [158]. In these applications, a tetrahedral mesh discretizes the shape or domain and provides support for background fields. Values at arbitrary points are obtained by interpolating data on the mesh. When element quality is poor (e.g., extreme dihedral angle), the interpolation error worsens, degrading accuracy and stability [291, 292]. Consequently, generating high-quality tetrahedral meshes has been a long-standing research focus.

For tetrahedralization, Delaunay-based methods are often preferred for practical performance [282, 293, 294], but they can be brittle in the presence of imperfect or noisy inputs. To address this,

TetWild [295, 296] introduced a robust tetrahedralization pipeline that attains mesh quality and performance comparable to Delaunay-based methods while tolerating imperfect surfaces. A separate line of work “stuffs” the interior of an isosurface using a space-tiling background grid that may be deformed or clipped [297–299]. Among them, Isosurface Stuffing [300] provides guarantees on element quality (e.g., bounded dihedral angles) and efficiency. However, none of these are differentiable, thus they are only used for post-processing after a surface has been reconstructed. In contrast, our approach is differentiable and can be integrated into end-to-end optimization pipelines.

8.2.2 Differentiable Mesh

Compared to implicit shape representations, meshes are widely regarded as more challenging to optimize in a differentiable setting because they couple continuous vertex positions with *discrete* connectivity. Changes in connectivity are inherently non-differentiable, which complicates gradient-based optimization. Consequently, several approaches fix the mesh topology and optimize only vertex positions [3, 4, 24, 160, 161]. More recently, probabilistic formulations have been suggested to model mesh connectivity, enabling dynamic topology updates during optimization [20, 289]. However, these methods produce only surface meshes, necessitating an additional post-processing stage to obtain a volumetric mesh.

Instead of optimizing a mesh directly, another approach extracts an isosurface from an implicit field whose parameters are explicitly defined [9, 11, 12, 162, 301]. Representative methods, such as DefTet [162], DMTet [9], and FlexiCubes [11], instantiate a voxel/tetrahedral lattice, assign SDF values (and other differentiable features) to its vertices, and recover the surface and enclosed volume via variants of classical isosurface extraction [22, 23, 138]. While effective for differentiable sur-

faces and volumes, these approaches rely on a topologically fixed background lattice, which limits the set of tetrahedral meshes to be represented. In contrast, our method updates grid topology by re-tetrahedralizing the moving vertices at each iteration, enabling a substantially broader class of meshes (fig. 8.2).

8.3 Approach

In this section, we first outline our overall approach to optimizing a tetrahedral mesh in a differentiable manner. Then, we introduce an extended version of the conventional Marching Tetrahedra (MT) algorithm to extract a volumetric mesh from a signed distance function (SDF). Finally, we provide our loss function to optimize the tetrahedral mesh for the given input modalities, including SDF and multi-view images.

8.3.1 Definition

We represent a shape by a set of vertices, each endowed with a spatial position and an SDF value. Vertex positions are the primary optimization variables; when a ground-truth SDF is unavailable (e.g., in multi-view reconstruction), the per-vertex SDF values can also be optimized. While this parameterization resembles prior work [9, 11], a key distinction is that we impose *no* pre-defined connectivity among vertices. This design aligns with DMesh-style approaches [20, 289], which avoid fixed lattices and capture connectivity via per-vertex features rather than a prescribed grid. Finally, we adopt the convention that negative SDF indicates the shape interior.

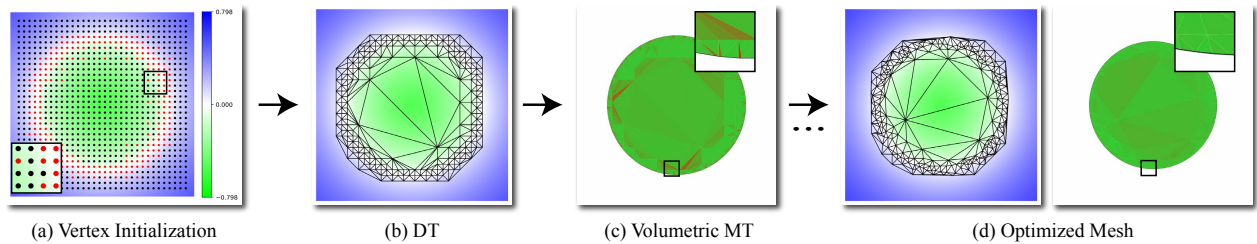


Figure 8.3: **Overview of our tetrahedral mesh optimization pipeline (illustrated in 2D).** (a) Initialize the vertex set: retain samples near the zero level set (red) and discard far-away samples (black). (b) Obtain the tetrahedral grid by computing a Delaunay triangulation (DT) of the retained vertices. (c) Using this grid and per-vertex SDF values, extract a volumetric tetrahedral mesh via a volumetric Marching Tetrahedra (MT) algorithm. Colors encode mesh quality; red triangles indicate poor aspect ratios. (d) Improve quality by minimizing losses defined on the extracted mesh and backpropagating to per-vertex attributes (positions and SDF values). After several iterations, the vertex features are optimized to produce an element-efficient, high-quality tetrahedral mesh.

8.3.2 Overview

We perform iterative, gradient-based optimization to get the optimal mesh from given inputs (see [fig. 8.3](#)).

Vertex Initialization. We initialize by placing a uniform voxel grid over the computational domain and using its lattice points as candidate vertices. Each vertex is assigned an SDF value: when a ground-truth SDF is available, we sample it directly; otherwise, we hold the grid fixed and run a brief pre-optimization to estimate per-vertex SDF values. We then prune vertices that are far from the zero level set, retaining only a narrow band of samples near the surface.

Delaunay Triangulation At each iteration, we compute a Delaunay triangulation (DT) of the current vertex set to obtain the background grid. This maintains valid element orientations and avoids the tangling or negative-volume inversions that can arise when deforming a fixed-topology grid.

Volumetric Marching Tetrahedra. We then apply our volumetric MT procedure ([Sec. 8.3.3](#)) to extract a volumetric mesh using the per-vertex SDF values and the tetrahedral grid from DT. The

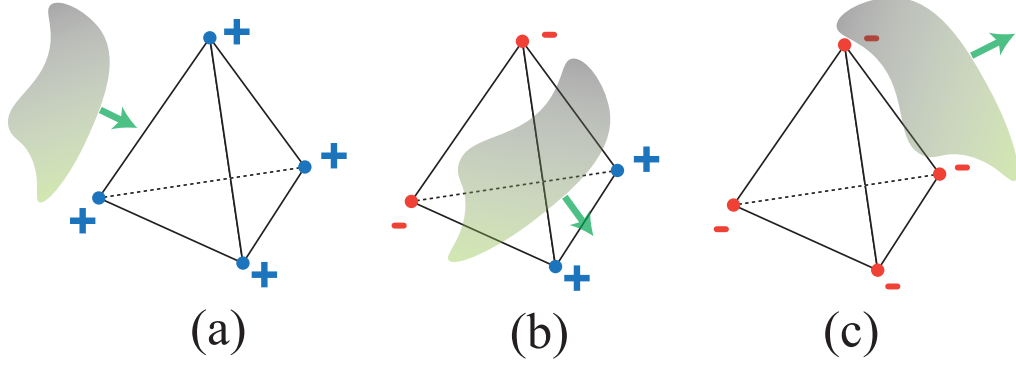


Figure 8.4: **Tetrahedron classification for volumetric mesh extraction.** Based on vertex signs, we can broadly classify a tetrahedron T into 3 cases. The sign of each vertex is labeled as + (outside) or - (inside), and a boundary surface patch of the target shape and its outward normal is rendered together. (a) When every vertex is outside, we can exclude T from our mesh. (b) When the boundary of the target shape intersects the tetrahedron, some vertices are outside while the others are inside. Then, we need to extract subset of T that is inside the target shape (see [fig. 8.5b](#)). (c) When every vertex is inside, we can include T in our mesh.

interior tetrahedra produced in this way typically satisfy the Delaunay property and are therefore well shaped ([fig. 8.3c](#)). Nonetheless, sliver elements [302] can still arise even under the Delaunay condition.

Evaluation. To suppress such pathologies while preserving geometric fidelity, we evaluate quality- and fidelity-oriented losses on the extracted mesh ([section 8.3.4](#)) and backpropagate gradients to the per-vertex attributes. Because we reconstruct tetrahedral grid at every step, the connectivity updates dynamically, enabling local topological changes in the grid throughout optimization.

8.3.3 Volumetric Marching Tetrahedra (MT)

The original MT algorithm [9, 23] is designed to extract surface meshes from an implicit field. To obtain a volumetric mesh, we extend MT to operate on volumes. Prior work on tetrahedralizing isosurfaces [300] targets a fixed, body-centered cubic lattice and is not directly compatible with our setting. We therefore devise a new, GPU-parallelizable procedure that is robust to connectivity changes.

We refer to this procedure as *volumetric MT*. As outlined in [algorithm 8](#) (Appendix), given a tetrahedralization $\mathbb{T}$ and an SDF S , we iterate over each tetrahedron $T^i \in \mathbb{T}$, compute the portion $\{ \mathbf{x} \in T^i \mid S(\mathbf{x}) \leq 0 \}$ inside the target shape, and tessellate this clipped region into a small set of tetrahedra. The union of these local volume tessellations yields the global volumetric mesh. Details of the clipping and local tetrahedralization steps are provided below.

8.3.3.1 Classification based on vertex SDF values

We start by evaluating the SDF values for the 4 vertices of T^i . Then, based on the combination of the vertices' signs, we can classify T^i into 3 cases shown in [fig. 8.4](#). Specifically, when every sign is positive, there is no inner part in T^i , so we discard it. Otherwise, if every sign is negative, it means that the whole T^i is inner part. Therefore, we include it in our mesh. When the boundary of the target shape intersects T^i , the signs would be mixed. In this case, we need to clip only the subset of T^i that is inside the target shape.

8.3.3.2 Clipping Algorithm

For T^i of which vertices have mixed signs, we approximate its inner part with a convex polyhedron V^i and decompose it into a set of tetrahedra. To get V^i , we can use linear interpolation along edges that intersect the target shape. That is, for an edge E that connects two vertices p_i and p_j with signed distance value of $s_i > 0$ and $s_j < 0$ respectively, we can approximate the point p_0 on E that is located on the boundary of the target shape as:

$$p_0 = \frac{s_i p_j - s_j p_i}{s_i - s_j}. \quad (8.1)$$

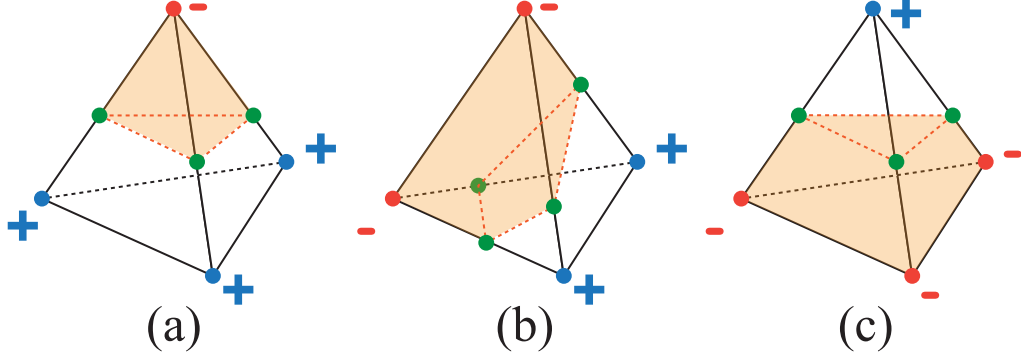


Figure 8.5: **Inner part approximation.** For each edge that connects vertices that have different signs, we can approximate the point on the iso-surface using linear interpolation, which are rendered in green. By connecting red inner vertices and these approximation points, we can get the orange convex polyhedron that approximates the inner part of the tetrahedron.

By connecting these interpolated points and the vertices with negative SDF values, we can get V^i as shown in [fig. 8.5](#).

In [fig. 8.6](#), we illustrate our tetrahedral decomposition process for V^i . To decompose V^i , we first compute the centroid C^i of V^i . This can be done by averaging the vertices of V^i , and it is guaranteed to be located inside V^i , as it is convex. Then, we compute a set of triangles $\mathbb{F}^i$ that triangulate the faces of V^i . Note that the faces of V^i are comprised of a face that approximates the iso-surface (F_{iso}^i), and the faces that are subsets of faces of T^i ($F_j^i |_{j=1,2,3,4}$). We can get the triangles $\mathbb{F}_{iso}^i$ for F_{iso}^i using the look-up table of the original MT algorithm [23]. For F_j^i , we can also take a similar approach based on look-up table to get triangles $\mathbb{F}_j^i$. Then, we get $\mathbb{F}^i$ by merging these triangle sets. Finally, for each triangle F in $\mathbb{F}^i$, we merge the triangle with C^i to generate a tetrahedron T^F , and add it to our mesh.

The quality of the decomposed tetrahedra depends mainly on the geometry of the clipped convex region. When this region is nearly planar (i.e., sliver-like), any valid local tetrahedralization tends to contain poorly shaped elements; in extreme cases, finite-precision effects can even place the computed centroid marginally outside the convex region. However, in practice, such cases are rare and are effec-

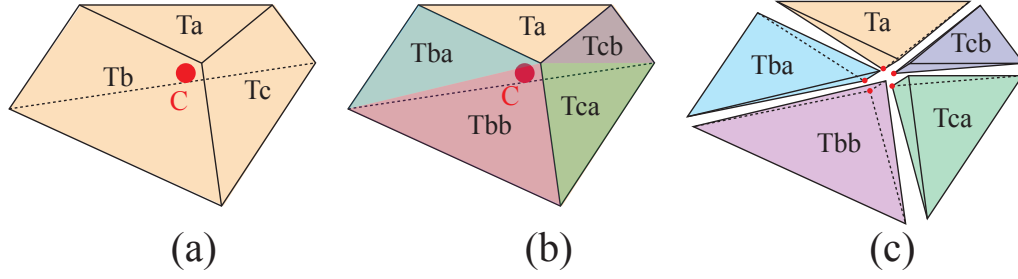


Figure 8.6: **Tetrahedral decomposition of a polyhedron.** (a) We decompose a polyhedron using a centroid C . (b) For each face T on the polyhedron, if it is not a triangle (e.g., T_b , T_c), we subdivide the face into a set of triangles (e.g., $T_b \rightarrow T_{ba}$, T_{bc}). (c) We can define a set of tetrahedra that compactly decomposes the polyhedron by connecting each triangle with C . The tetrahedra not visible from the front are not rendered.

tively suppressed by our mesh quality loss ([section 8.3.4.2](#)), which penalize ill-shaped configurations and steer the optimization toward well-conditioned elements.

Before moving on, we would like to note that the volumetric MT procedure is massively parallelizable: the operation that extracts the interior of each tetrahedron is independent across elements, and the lookup-table triangulation of polygonal faces (and subsequent local tetrahedralization of the clipped polyhedron) reduces to simple index remapping. Consequently, the algorithm admits efficient GPU implementation with no inter-element data dependencies.

8.3.4 Loss Formulation

After obtaining the mesh M using the volumetric MT algorithm, we evaluate it with a loss function L . There are various measures to evaluate the mesh, but here we mainly consider the losses for geometric fidelity and mesh quality.

8.3.4.1 Geometric Fidelity

We can employ different loss functions to measure the geometric fidelity of M based on the input modality. Here we discuss loss functions for SDF and image inputs.

SDF. We first sample N points on M 's surface. Then, for every sample point, we evaluate its signed distance value using the given SDF, and get their average absolute value to compute distance loss:

$$L_{\text{dist}}(M) = \frac{1}{N} \sum_{i=1}^N |s_i|, \quad (8.2)$$

where s_i is the signed distance value of i -th point p_i .

To further enhance the accuracy, we compute normal loss from the sampled points. Specifically, for each sampled point, we compute the normal direction $\vec{d}_f$ of the face that the point is sampled from, and compare it with the ground truth normal direction $\vec{d}_{gt}$ estimated from SDF. Note that we can estimate $\vec{d}_{gt}$ easily by computing the first-order derivative of the SDF with respect to the input point position. Then the normal loss is defined as:

$$L_{\text{normal}}(M) = \frac{1}{N} \sum_{i=1}^N 1 - |\vec{d}_{f,i} \cdot \vec{d}_{gt,i}|. \quad (8.3)$$

Images. When multi-view images are given as input, we render M with a differentiable renderer [24] and compare with the given images. For comparison, we use L_1 loss:

$$L_{\text{image}}(M) = \frac{1}{N} \sum_{i=1}^N |\mathcal{L}_i^{gt} - \mathcal{L}_i^M|, \quad (8.4)$$

where N is the number of images to compare per batch, $\mathcal{L}_i^{gt}$ is the i -th given image, and $\mathcal{L}_i^M$ is the i -th

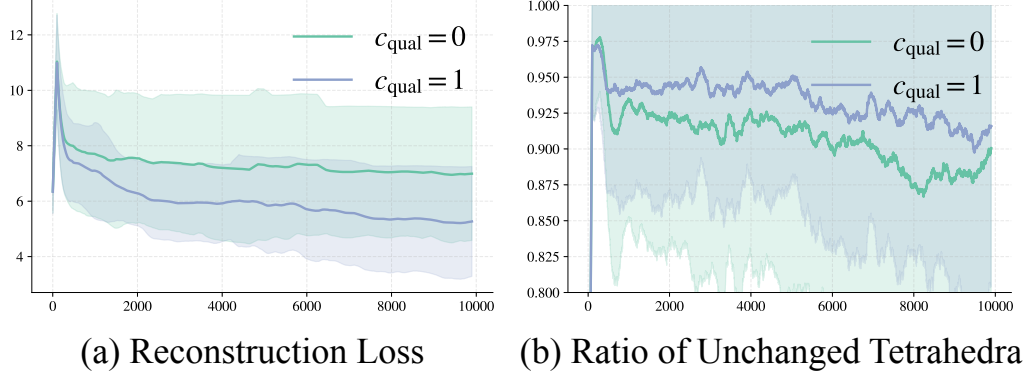


Figure 8.7: **Effect of quality loss in differentiability.** (a) We can attain better reconstruction loss when we use quality loss. (b) This is because it contributes to suppressing discontinuous topological changes, which could harm the optimization process.

rendered image of our mesh.

8.3.4.2 Mesh Quality

Downstream applications like physics simulation often require a mesh to satisfy certain quality constraints, and volume-length measure is one of the widely used metrics to measure the mesh quality [282]. For a tetrahedron T , its volume-length measure $Q(T)$ is computed as:

$$Q(T) = \frac{V}{l_{rms}^3}, \quad (8.5)$$

where V is the volume of T , and l_{rms} is root-mean-squared length of T 's 6 edges. This value reaches the maximum value of $\frac{1}{6\sqrt{2}}$ when T is ideal.

While a tetrahedron in Delaunay triangulation usually has good quality, it could become a sliver, which has a very small dihedral angle. The volume-length measure detects such slivers and allows us to remove them. Note that when the minimum dihedral angle is near zero, the volume-length measure becomes zero. Also, compared to the other measures like aspect ratio, it is easier and more stable to

compute, because we do not have to compute the circumradius of a tetrahedron [282]. Moreover, this loss also contributes to improving the surface mesh quality, as shown in [fig. 8.1](#).

Using this volume-length measure, we compute the quality loss of M as follows:

$$L_{\text{quality}}(M) = \frac{1}{|\mathbb{T}|} \sum_{T \in \mathbb{T}} \frac{1}{6\sqrt{2}} - Q(T). \quad (8.6)$$

Differentiability Beyond improving element shape, the quality loss stabilizes gradient-based optimization by reducing discrete connectivity changes that introduce discontinuity. Note that the mesh we get from our pipeline is fully differentiable with respect to the vertex attributes when the tetrahedral grid is fixed, and the sign of the vertices remains the same. However, discontinuities could arise from 1) DT connectivity changes and 2) SDF sign flips in the lookup-based volumetric MT. Due to these factors, the gradients that we get also become discontinuous, which could undermine the optimization process.

However, despite these potential discontinuities, we find the gradients are still usable in practice. This is because SDF interpolation, together with the typically well-shaped elements produced by DT, reduces abrupt changes in the extracted mesh. In particular, even when the tetrahedral grid’s connectivity changes, appropriate per-vertex SDF values keep the interpolated surface nearly unchanged when the tetrahedral elements are well-shaped.

To prove this, we render reconstruction loss curves for the SDF-conditioned mesh optimization task ([section 8.4.2](#)), using 1,000 randomly initialized vertices over five seeds and a SDF representing unit sphere ([fig. 8.7](#)). The loss decreases even without a quality term ($c_{\text{quality}} = 0$), but convergence is suboptimal due to slivers and sign flips. Enabling the quality loss ($c_{\text{quality}} = 1$) yields faster loss reduction and a higher per-iteration fraction of preserved tetrahedra. This is because the quality loss

effectively removes slivers in our tetrahedral mesh, and suppresses sign flips of the vertices (as the tetrahedral quality worsens before sign flip happens).

8.3.4.3 Total Loss

We compute our total loss as:

$$L(M) = c_{\text{fidelity}} \cdot L_{\text{fidelity}}(M) + c_{\text{quality}} \cdot L_{\text{quality}}(M), \quad (8.7)$$

where L_{fidelity} is the geometric fidelity loss dependent on the input modality. Exact coefficient values are detailed in the next section.

8.4 Experimental Results

In this section, we present experimental results demonstrating our method’s ability to produce high-quality, element-efficient tetrahedral meshes. We first report optimization results conditioned on known signed distance fields (SDFs), followed by multi-view 3D reconstruction results. Our implementation is in Python with PyTorch [90]. Delaunay triangulation is performed using CGAL [303], leveraging its CPU parallelization. All of our experiments were conducted on a workstation with an Intel® Xeon® W-2255 (3.70 GHz) CPU and a single NVIDIA RTX A5000 GPU.

8.4.1 Metrics

To evaluate the geometric accuracy of the extracted tetrahedral mesh, we select its boundary and compute Chamfer Distance (**CD**), F1 score (**F1**), and Normal Consistency (**NC**) to the ground truth mesh [19]. To assess the mesh efficiency, we count the total number of tetrahedra (**#T**). Finally,

Table 8.1: **Quantitative Comparison of Extracted Mesh**

Method	CD($\times 10^{-5}$) $\downarrow$	F1 $\uparrow$	NC $\uparrow$	#T(Mil.) $\downarrow$	BT(%) $\downarrow$
MC(256) [22]	1.975	0.674	0.993	-	-
MT(128) [23]	5.225	0.641	0.981	1.469	12.22
Ours	1.984	0.670	0.992	0.754	1.204

to measure the mesh quality, we compute the ratio of tetrahedra whose minimum dihedral angle is smaller than 10° in the mesh (**BT**).

8.4.2 SDF Conditioned Mesh Optimization

Dataset For this experiment, we use 256 models from the ABC dataset [304], consisting primarily of mechanical parts (fig. 8.8). We select this dataset to demonstrate our ability to recover sharp geometric features. As preprocessing, each mesh was converted to a SDF at a resolution of 384^3 [305]. For evaluation, a reference surface was extracted from the SDF using Marching Cubes (MC) on 384^3 voxel grid.

Baselines To evaluate the geometrical accuracy of the mesh, we employ the standard iso-surface extraction algorithms for comparison. They include Marching Cubes (MC) [22] and Marching Tetrahedra (MT) [23]. For MC, we used grid resolution of 256^3 . For MT, we simply used our volumetric MT algorithm, as the surface mesh we get from the algorithm is the same as the vanilla MT [section 8.3.3.2](#). We initialized vertices for MT using a voxel grid, of which resolution is 128^3 , and built tetrahedral grid using DT. Note that the resolution was set different between MC and MT to produce similar number of faces on the surface mesh.

Our Settings We initialize the vertex set on the 128^3 , followed by pruning. We then optimize for 1,000 iterations using Adam [260] with a learning rate of 10^{-4} . We set the loss weights to $c_{\text{dist}} = 10^2$,

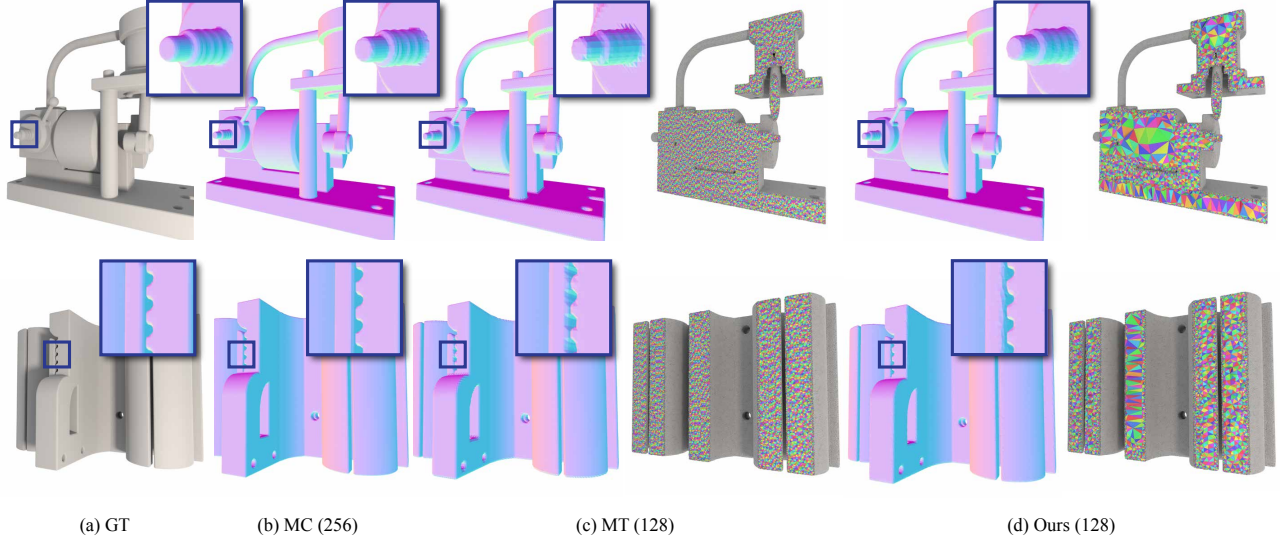


Figure 8.8: **Qualitative comparison of the volumetric mesh representing mechanical structures.** Using our method, we can attain geometrically accurate mesh comparable to MC [22] that used $2\times$ higher grid resolution, while the baseline MT [23] is noticeably less accurate. Our mesh is comparably more element-efficient than MT, as ours prunes away unnecessary vertices.

$c_{\text{normal}} = 10$, and $c_{\text{quality}} = 1$.

Comparisons After optimization, our method produces an element-efficient, high-quality tetrahedral mesh without sacrificing accuracy (fig. 8.8). Compared with the baseline MT, our mesh preserves sharp, fine-scale features and closely matches the detail achieved by MC, while using substantially fewer elements. The reduction in element count follows from pruning redundant vertices. These qualitative observation aligns well with the quantitative results in table 8.1: our geometric accuracy improves markedly over MT and is only marginally below MC. In terms of efficiency, our meshes contain 49% fewer tetrahedra than MT. Moreover, the fraction of low-quality elements was significantly lower than in MT. Overall, these results confirm that our differentiable framework recovers high-quality, element-efficient tetrahedral meshes directly from SDF.

Table 8.2: **Quantitative comparison of the multi-view reconstruction results.** For each metric, the better result is highlighted with color.

Method	Metric	Armadillo	Bimba	Botijo	Bumpy Sphere	Bunny	Hand	Lion	Gargoyle	Knot
FlexiCubes [11]	CD (10^{-5})↓	1.855	2.218	1.473	3.167	2.470	1.583	2.566	2.816	4.024
	F1↑	0.623	0.554	0.704	0.432	0.518	0.679	0.502	0.469	0.358
	NC↑	0.973	0.991	0.992	0.996	0.995	0.989	0.985	0.974	0.984
	# T. (Mil)↓	0.530	1.282	0.521	0.972	1.540	0.574	1.231	0.959	1.266
	# BT (%).↓	4.751	2.707	3.949	2.123	2.465	3.849	2.866	3.598	4.866
Ours	CD	2.167	2.712	1.475	3.112	2.569	2.111	3.074	3.065	3.788
	F1	0.624	0.545	0.699	0.435	0.507	0.684	0.507	0.468	0.367
	NC	0.977	0.988	0.990	0.996	0.996	0.986	0.986	0.975	0.982
	# T	0.521	0.828	0.403	0.553	1.165	0.474	0.752	0.742	1.174
	# BT (%).	4.460	2.343	1.781	2.781	3.143	2.201	2.982	5.971	4.589

8.4.3 Mesh Reconstruction from Multi-View Images

Dataset For evaluation, we used 9 representative meshes from the dataset collected by [306]. For each mesh, we randomly sampled 8 viewpoints for each iteration, and rendered mask, depth, and normal maps of resolution (2048, 2048) for computing the reconstruction loss.

Baselines We mainly compare our reconstruction results with those from FlexiCubes [11]. We set the grid resolution to 96^3 , and optimized for 3,000 iterations with the default settings to get the best results. Also, for the fair comparison, we tried to use FlexiCube’s protocols as closely as possible, only replacing the shape representation with ours.

Our Settings We used the same grid resolution of 96^3 as the FlexiCubes, and optimized for the same 3,000 iterations. However, as we do not have ground truth SDF in this case, we ran preliminary optimization for vertex-wise SDF values with fixed grid for the first 1,000 steps (section 8.3.2). Then we pruned redundant vertices and optimized both vertex positions and SDF values for another 2,000 steps. We used Adam optimizer [260], and set the learning rate to 10^{-2} for the preliminary optimization, and 10^{-3} for the next phase. For the loss weights, we set $c_{\text{image}} = 1$, and $c_{\text{quality}} = 10^{-3}$.

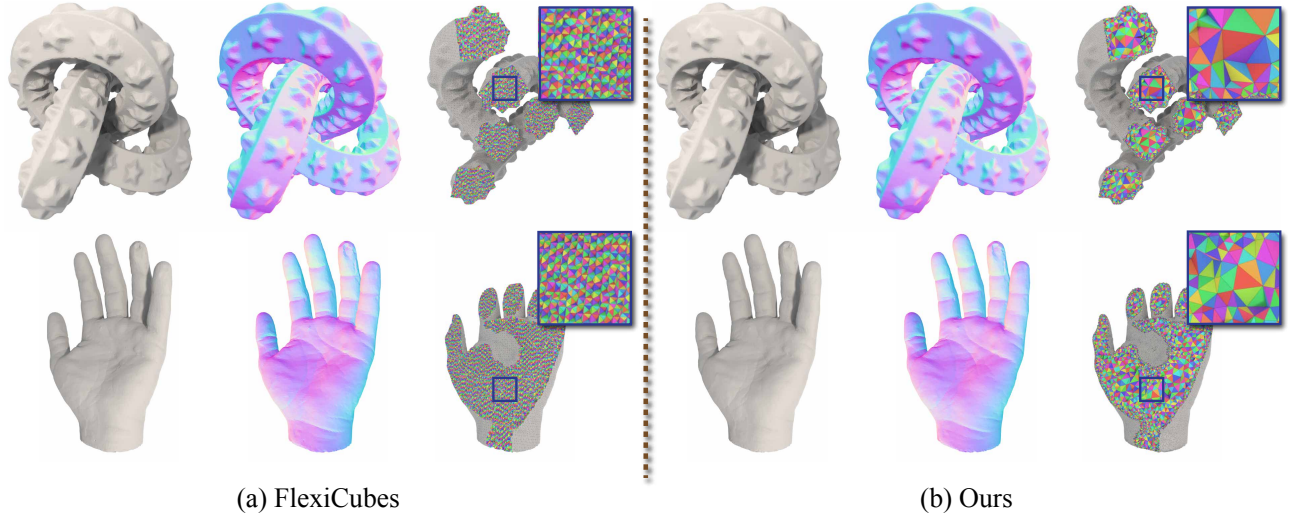


Figure 8.9: **Qualitative comparison of the reconstruction results from multi-view images (From top: Knot, Hand, and Bunny)** Compared to FlexiCubes [11], our method reconstructs much more element-efficient mesh, without losing visual details and mesh quality.

Comparisons As shown in [fig. 8.2](#) and [fig. 8.9](#), our reconstruction results are more element-efficient than FlexiCubes while preserving comparable near-surface detail. The quantitative results in [table 8.2](#) corroborate this observation: although FlexiCubes achieves a slightly lower Chamfer Distance (CD), the remaining accuracy metrics were almost the same. On average, our method produced 26% fewer tetrahedra than FlexiCubes, and the fraction of low-quality tetrahedra is comparable between the two. Specifically, note that the reduction in element count varies across inputs. Because our pruning retains only a narrow SDF band around the zero level set, shapes with greater interior “thickness” discard more interior vertices and thus see a larger reduction; see the Bumpy Sphere in [fig. 8.2](#) and the Knot in the top row of [fig. 8.9](#). Taken together, these results indicate that our approach yields more element-efficient meshes without sacrificing geometric fidelity or element quality.

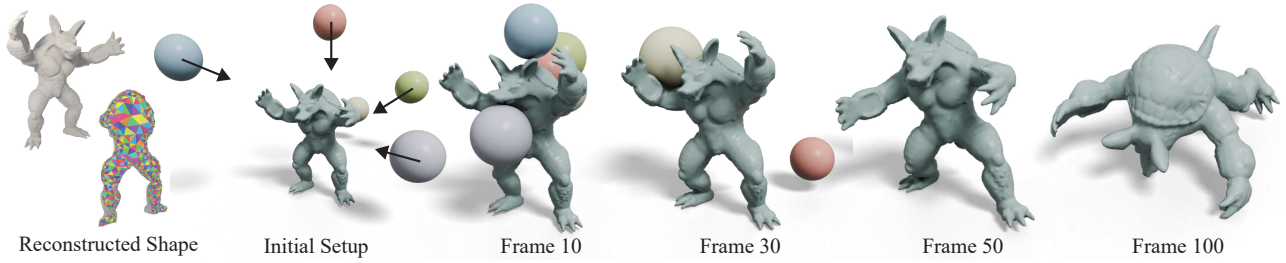


Figure 8.10: **Physics simulation using reconstructed Armadillo.** We pass the Delaunay-triangulated output of our method directly to an elastic FEM simulator with contact. Rigid balls are dropped and collide with the deformable body, producing contact-rich behavior. The full animation is provided in the supplementary video.

8.4.4 Validation with Physics Simulation

We validate downstream usability by feeding the output of our method, i.e., a solver-ready tetrahedral mesh, directly into an elastic FEM simulator. Using the Armadillo reconstructed by our pipeline, we simulate multiple rigid balls that bounce and collide with the deformable body. The mesh runs without remeshing or manual cleanup and exhibits stable, contact-rich deformations; results are shown in [fig. 8.10](#) and full animation is in Supplementary Video.

8.5 Conclusion

In this work, we introduced a gradient-based framework for tetrahedral-mesh optimization. Unlike prior differentiable mesh methods, we do not enforce a topologically fixed background lattice during optimization. Instead, vertices move freely in the space, and at each iteration we reconstruct a well-posed tetrahedral grid via Delaunay triangulation (DT). This dynamic connectivity allows us to represent a broader class of tetrahedral meshes than fixed-grid approaches. Furthermore, we presented a GPU-friendly volumetric Marching Tetrahedra (MT) algorithm to extract the volumetric mesh efficiently from the grid. Together, these components yield element-efficient, high-quality tetrahedral

meshes from SDFs and multi-view images without sacrificing geometric detail.

Limitations However, the main bottleneck of our approach is in computational cost. Our computational cost is more expensive than the baseline methods. For the multi-view reconstruction experiments, our method required $1.67\times$ more wall-clock time in average than FlexiCubes, even though they optimized for the same number of iterations. This is because we have to run DT at every optimization step. Also, for the SDF conditioned mesh optimization, ours definitely take longer time than the baselines, because they are not optimization-based methods.

Future Work To address this limitation, a learning-based approach is promising for generating element-efficient tetrahedral meshes from input data without optimization. For example, one could train a neural network to predict per-vertex attributes (i.e., positions and SDF values) that yield high-quality meshes for the given inputs. Because our pipeline is fully differentiable, it can be integrated seamlessly into such learning frameworks.

Second, our framework can be extended to reconstruct element-efficient meshes with estimated physical properties from real-world images or videos, enabling direct use in physics simulation and robotics training. At present, we demonstrate simulation with the multi-view–reconstructed Armadillo (fig. 8.10). In future work, we aim to reconstruct textured meshes that can be simulated efficiently thanks to their low element counts.

Taken together, this chapter completes the geometric arc of the thesis by extending gradient-based optimization from surface representations to volumetric domains with adaptive topology. Across dynamics, control, and geometry, a unifying theme emerges: gradients serve as a general mechanism for reasoning about input–output sensitivity, whether the underlying system is continuous, discrete, or topologically evolving. In the final chapter, we reflect on this unifying perspective and discuss how

gradient-informed modeling can support scalable optimization, learning, and simulation for complex physical systems.

Part V

Conclusion

Chapter 9: Conclusion

9.1 Summary of Results

To summarize, my work has focused on developing specialized reinforcement learning algorithms for continuous and differentiable physical environments, and designing differentiable formulations for kinematics and geometry, which could be used for efficient optimization and learning. My specific contributions can be summarized as follows.

Gradient Informed Proximal Policy Optimization [307] In this project, we introduced a novel policy learning method that integrates analytical gradients from differentiable environments with the Proximal Policy Optimization (PPO) algorithm. Our proposed approach outperforms baseline algorithms in various scenarios, such as function optimization, physics simulations, and traffic control environments. Code can be found online: <https://github.com/SonSang/gippo>.

Time-Aware World Model for Adaptive Prediction and Control [308] In this work, we proposed the Time-Aware World Model (TAWM), a model-based approach that explicitly incorporates temporal dynamics. Empirical evaluations show that TAWM consistently outperforms conventional models across varying observation rates in a variety of control tasks, using the same number of training samples and iterations. Our code can be found online at: <https://github.com/anh-nn01/Time-Aware-World-Model>.

Differentiable Hybrid Traffic Simulation [239] In this project, we introduced a novel differentiable hybrid traffic simulator, which simulates traffic using a hybrid model of both macroscopic and microscopic models and can be directly integrated into a neural network for traffic control and flow

optimization. Thanks to the differentiability, our simulator can provide more efficient and scalable solutions for complex learning and control problems posed in traffic engineering than other existing algorithms. Code is available at: <https://github.com/SonSang/diff-hybrid-traffic-sim>.

Gradient-based Trajectory Optimization with Parallelized Differentiable Traffic Simulation [309]

Here we presented a parallelized differentiable traffic simulator based on the Intelligent Driver Model (IDM), a car-following framework that incorporates driver behavior as key variables. With the capability to simulate up to 2 million vehicles in real time, the system is scalable for large-scale trajectory optimization. The code is publicly available at: <https://github.com/SonSang/diffidm>.

DMesh: A Differentiable Mesh Representation [20] In this work, we presented a differentiable representation, DMesh, for general 3D triangular meshes. DMesh considers both the geometry and connectivity information of a mesh in a differentiable manner. DMesh can be used to represent meshes of various topology in a differentiable way, and allows us to reconstruct the mesh under various observations, such as point clouds and multi-view images using gradient-based optimization. The code is available at: <https://github.com/SonSang/dmesh>.

DMesh++: An Efficient Differentiable Mesh for Complex Shapes [289] We introduce a new differentiable mesh processing method that addresses computational challenge of DMesh and efficiently handles meshes with intricate structures. Our method reduces time complexity from $O(N)$ to $O(\log N)$ and requires significantly less memory than DMesh. Building on this innovation, we present a reconstruction algorithm capable of generating complex 2D and 3D shapes from point clouds or multi-view images. The code is publicized at: <https://github.com/SonSang/dmesh2>.

Differentiable Delaunay Tetrahedral Mesh Generation In this work, we introduce a differentiable framework for tetrahedral mesh optimization. Given a set of vertices, each with an associated signed distance function (SDF) value, we construct a Delaunay tetrahedralization (DT) and extract a volumetric mesh via a volumetric extension of the Marching Tetrahedra (MT) algorithm. We validate our approach by reconstructing geometrically accurate, high-quality tetrahedral meshes from both prescribed SDF fields and multi-view images. Compared to baselines, our method produces a more element-efficient, high-quality mesh with upto 49% fewer elements without losing fidelity. The code is publicized at: <https://github.com/SonSang/difftet>.

9.2 Future Work

Looking ahead, the field can be advanced by further integrating reinforcement learning and differentiable modeling toward what might be called *physical AI*: agents that reason, predict, and act in the physical world by leveraging explicit models of geometry and dynamics. Building on the foundations laid in this thesis, several promising directions emerge, with an eye toward concrete real-world systems such as autonomous robots, interactive simulators, and large-scale digital twins for transportation and built environments.

World-model-based RL for physical intelligence A first direction is to further develop reinforcement learning algorithms that make more systematic use of world models in continuous, physics-based environments. The current work combines analytical gradients and policy optimization, but largely treats model-based and model-free ideas separately. Future research can design algorithms that tightly couple learned or differentiable world models with policy optimization, enabling agents to plan, imagine counterfactual futures, and adapt online to changing conditions. This includes learning

uncertainty-aware world models that can reason over variable time scales, integrating differentiable simulators directly into model-based planning loops, and scaling such methods to high-dimensional robotic systems. In practical terms, such algorithms could allow legged robots to adapt their gaits to novel terrains, manipulators to replan under contact uncertainty in cluttered environments, and autonomous vehicles to negotiate dense, interactive traffic more safely and efficiently.

Differentiable physical and geometric simulators A second direction is to broaden the scope and fidelity of differentiable physics and geometry simulation. The simulators in this thesis already demonstrate that exact or approximate gradients can substantially accelerate optimization and control, but they are specialized to particular domains, such as traffic or meshes. Going forward, a key goal is to construct a more unified family of differentiable simulators that cover rigid and deformable bodies, contact-rich interactions, and complex geometries under a common framework. Such simulators should be efficient enough for large-scale training, yet physically faithful enough to support reliable gradient-based design and control across robotics, graphics, and engineering applications. In the long term, such tools could form the backbone of high-fidelity digital twins—for example, simulating fleets of robots in warehouses, optimizing traffic flow in entire cities, or co-designing mechanical structures and control policies for next-generation robotic systems.

Perceptual models that recover geometry, topology, and physical properties Finally, an important direction is to develop AI systems that can infer rich physical structure from raw observations, such as images or videos. Beyond reconstructing geometry efficiently, a key goal is to recover material parameters, constraints, and forces in a way that is both consistent with physical laws and useful for downstream control. Of particular interest are models that, like humans, form a detailed understand-

ing of topology: distinguishing connectivity, holes, and handles, and reasoning about how objects can deform or break. By combining differentiable geometric representations (such as DMesh and differentiable tetrahedral meshes) with physics-informed world models, future systems could yield agents that not only see the world, but also understand how it is put together and how it will respond to their actions. Such perceptual capabilities are essential for robots that must manipulate everyday objects safely, for simulators that support reliable virtual prototyping, and for immersive AR/VR systems that require physically plausible interactions with complex 3D environments.

Part VI

Appendix

Appendix A: Gradient Informed Proximal Policy Optimization

A.1 Proofs

A.1.1 Proof of Lemma 2.3.2

By Definition 2.3.1, $\pi_\theta(s, a)$ should satisfy the following condition:

$$\begin{aligned} \int_{T_\epsilon} q(\epsilon) d\epsilon &= \int_{T_a} \pi_\theta(s, a) da \\ &= \int_{T_\epsilon} \pi_\theta(s, g_\theta(s, \epsilon)) \cdot \left| \det\left(\frac{\partial g_\theta(s, \epsilon)}{\partial \epsilon}\right) \right| d\epsilon \quad (\text{by change of variable}). \end{aligned}$$

We denote the set of continuous distributions that satisfy this condition as P . That is,

$$P_\theta(s, \epsilon) \in P \text{ if } \int_{T_\epsilon} q(\epsilon) d\epsilon = \int_{T_\epsilon} P_\theta(s, \epsilon) \cdot \left| \det\left(\frac{\partial g_\theta(s, \epsilon)}{\partial \epsilon}\right) \right| d\epsilon.$$

Clearly, $\bar{P}_\theta(s, \epsilon) = q(\epsilon) \cdot \left| \det\left(\frac{\partial g_\theta(s, \epsilon)}{\partial \epsilon}\right) \right|^{-1}$ is an element of P , and is well defined, because of Equation 2.3. In fact, $\bar{P}_\theta(s, \epsilon)$ is the only element of P , as the above condition should be met for any $T_\epsilon \subset \mathbb{R}^n$. That is, if there was another distribution $\tilde{P}_\theta(s, \epsilon) \in P$,

$$\int_{T_\epsilon} (\bar{P}_\theta(s, \epsilon) - \tilde{P}_\theta(s, \epsilon)) \cdot \left| \det\left(\frac{\partial g_\theta(s, \epsilon)}{\partial \epsilon}\right) \right| d\epsilon = 0, \forall T_\epsilon \subset \mathbb{R}^n.$$

Since $\tilde{P}_\theta$ is different from $\bar{P}_\theta$, there exists an open set $\hat{T}_\epsilon$ where $\tilde{P}_\theta < \bar{P}_\theta$. Then

$$\int_{\hat{T}_\epsilon} (\bar{P}_\theta(s, \epsilon) - \tilde{P}_\theta(s, \epsilon)) \cdot \left| \det\left(\frac{\partial g_\theta(s, \epsilon)}{\partial \epsilon}\right) \right| d\epsilon > 0,$$

which is a contradiction. Therefore, $\pi_\theta(s, a) = \pi_\theta(s, g_\theta(s, \epsilon)) = \bar{P}_\theta(s, \epsilon)$. The inverse holds trivially.

A.1.2 Proof of Lemma 2.4.2

First, we show that f is an injective function by showing that the determinant of Jacobian $\frac{\partial f}{\partial a}$ is always positive:

$$\begin{aligned} \det\left(\frac{\partial f}{\partial a}\right) &= \det(I + \alpha \cdot \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a)) \\ &= \prod_{i=1}^n (1 + \alpha \cdot \lambda_i(s, a)) > 0 \quad (\because |\alpha| < \frac{1}{\max_{(s,a)} |\lambda_1(s, a)|}), \end{aligned}$$

where $\lambda_i(s, a)$ are the eigenvalues of $\nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a)$ sorted in ascending order.

Then, for an arbitrary open set of action $T \subset A$, $\pi_\alpha(s, \cdot)$ selects $\tilde{T} = f(T) \subset A$ with the same probability that the original policy $\pi_{\bar{\theta}}(s, \cdot)$ selects T .

$$\begin{aligned} \int_{\tilde{T}} \pi_\alpha(s, \tilde{a}) d\tilde{a} &= \int_T \frac{\pi_{\bar{\theta}}(s, a)}{|\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))|} \cdot \left| \det\left(\frac{\partial f}{\partial a}\right) \right| da \\ &= \int_T \frac{\pi_{\bar{\theta}}(s, a)}{|\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))|} \cdot \left| \det(I + \alpha \cdot \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a)) \right| da \\ &= \int_T \pi_{\bar{\theta}}(s, a) da. \end{aligned}$$

Therefore, $\pi_\alpha(s, \cdot)$ is a valid probability distribution as $\pi_{\bar{\theta}}(s, \cdot)$.

A.1.3 Proof of Proposition 2.4.3

For a given state s , we can estimate the (approximate) expected value of the state under π_α as follows.

$$\begin{aligned}
& \int_{\tilde{a}} \pi_\alpha(s, \tilde{a}) A_{\pi_{\bar{\theta}}}(s, \tilde{a}) d\tilde{a} \\
&= \int_a \frac{\pi_{\bar{\theta}}(s, a)}{|\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))|} A_{\pi_{\bar{\theta}}}(s, a + \alpha \nabla_a A_{\pi_{\bar{\theta}}}(s, a)) |\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))| da \\
&\approx \int_a \pi_{\bar{\theta}}(s, a) [A_{\pi_{\bar{\theta}}}(s, a) + \alpha \|\nabla_a A_{\pi_{\bar{\theta}}}(s, a)\|^2] da \\
&= \int_a \pi_{\bar{\theta}}(s, a) A_{\pi_{\bar{\theta}}}(s, a) da + \alpha \int_a \pi_{\bar{\theta}}(s, a) \|\nabla_a A_{\pi_{\bar{\theta}}}(s, a)\|^2 da.
\end{aligned}$$

Therefore, we can see the following holds:

$$\begin{aligned}
L_{\pi_{\bar{\theta}}}(\pi_\alpha) &= \int_s \rho_{\pi_{\bar{\theta}}}(s) \int_{\tilde{a}} \pi_\alpha(s, \tilde{a}) A_{\pi_{\bar{\theta}}}(s, \tilde{a}) d\tilde{a} \\
&\approx \eta(\pi_{\bar{\theta}}) + \alpha \int_s \rho_{\pi_{\bar{\theta}}}(s) \int_a \pi_{\bar{\theta}}(s, a) \|\nabla_a A_{\pi_{\bar{\theta}}}(s, a)\|^2 da.
\end{aligned}$$

Since $\pi_{\bar{\theta}}(s, a)$ and $\|\nabla_a A_{\pi_{\bar{\theta}}}(s, a)\|^2$ are positive, $L_{\pi_{\bar{\theta}}}(\pi_\alpha)$ is greater or equal to $\eta(\pi_{\bar{\theta}})$ when $\alpha > 0$.

On the contrary, when $\alpha < 0$, $L_{\pi_{\bar{\theta}}}(\pi_\alpha)$ is smaller or equal to $\eta(\pi_{\bar{\theta}})$. Since this argument builds upon local approximation, it holds only when $|\alpha| \ll 1$.

A.1.4 Proof of Lemma 2.4.4

When $a = g_{\bar{\theta}}(s, \epsilon)$,

$$\begin{aligned}
g_{\alpha}(s, \epsilon) &= g_{\bar{\theta}}(s, \epsilon) + \alpha \cdot \nabla_{g_{\bar{\theta}}(s, \epsilon)} A_{\pi_{\bar{\theta}}}(s, g_{\bar{\theta}}(s, \epsilon)) \\
\Rightarrow \frac{\partial g_{\alpha}(s, \epsilon)}{\partial \epsilon} &= \frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon} + \alpha \cdot \nabla_{g_{\bar{\theta}}(s, \epsilon)}^2 A_{\pi_{\bar{\theta}}}(s, g_{\bar{\theta}}(s, \epsilon)) \cdot \frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon} \\
&= \left(I + \alpha \cdot \nabla_{g_{\bar{\theta}}(s, \epsilon)}^2 A_{\pi_{\bar{\theta}}}(s, g_{\bar{\theta}}(s, \epsilon)) \right) \cdot \frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon} \\
\Rightarrow \det \left(\frac{\partial g_{\alpha}(s, \epsilon)}{\partial \epsilon} \right) &= \det \left(I + \alpha \cdot \nabla_{g_{\bar{\theta}}(s, \epsilon)}^2 A_{\pi_{\bar{\theta}}}(s, g_{\bar{\theta}}(s, \epsilon)) \right) \cdot \det \left(\frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon} \right) \\
&= \det \left(I + \alpha \cdot \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a) \right) \cdot \det \left(\frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon} \right).
\end{aligned}$$

Since $\left| \det \left(\frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon} \right) \right| > 0$ (Equation 2.3), we can divide both sides of the above equation with $\det \left(\frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon} \right)$ and prove Lemma 2.4.4.

A.1.5 Proof of Lemma 2.4.6

From Equation 2.11,

$$\nabla_a \hat{A}_{\pi_{\theta}}(s_t, a_t) = \frac{1}{2} \mathbb{E}_{s_t, a_t, \dots \sim \pi_{\theta}} \left[\sum_{k=t}^{\infty} \gamma^k \frac{\partial r(s_k, a_k)}{\partial a_t} \right],$$

which can be rewritten using g_{θ} as follows.

$$\nabla_a \hat{A}_{\pi_{\theta}}(s_t, a_t) = \frac{1}{2} \mathbb{E}_{s_t, \epsilon_t, \dots \sim q} \left[\sum_{k=t}^{\infty} \gamma^k \frac{\partial r(s_k, g_{\theta}(s_t, \epsilon_t))}{\partial g_{\theta}(s_t, \epsilon_t)} \right]. \quad (\text{A.1})$$

Then, by differentiating Equation 2.10 with respect to θ after plugging in $\hat{A}$, we can get following

relationship:

$$\begin{aligned}
\frac{\partial L(\theta)}{\partial \theta} &= \mathbb{E}_{s_0, \epsilon_0, \dots, \sim q} \left[\sum_{t=0}^{\infty} 2 \cdot \frac{\partial g_{\theta}(s_t, \epsilon_t)}{\partial \theta} \cdot (g_{\theta}(s_t, \epsilon_t) - g_{\alpha}(s_t, \epsilon_t)) \right] \\
&= \mathbb{E}_{s_0, \epsilon_0, \dots, \sim q} \left[\sum_{t=0}^{\infty} 2 \cdot \frac{\partial g_{\theta}(s_t, \epsilon_t)}{\partial \theta} \cdot (g_{\theta}(s_t, \epsilon_t) - (g_{\theta}(s_t, \epsilon_t) + \nabla_a \hat{A}_{\pi_{\theta}}(s_t, a_t))) \right] \\
&(\because \alpha = 1) \\
&= \mathbb{E}_{s_0, \epsilon_0, \dots, \sim q} \left[\sum_{t=0}^{\infty} -2 \cdot \frac{\partial g_{\theta}(s_t, \epsilon_t)}{\partial \theta} \cdot \nabla_a \hat{A}_{\pi_{\theta}}(s_t, a_t) \right] \\
&= \mathbb{E}_{s_0, \epsilon_0, \dots, \sim q} \left[\sum_{t=0}^{\infty} -2 \cdot \frac{\partial g_{\theta}(s_t, \epsilon_t)}{\partial \theta} \cdot \frac{1}{2} \cdot \sum_{k=t}^{\infty} \gamma^k \frac{\partial r(s_k, g_{\theta}(s_t, \epsilon_t))}{\partial g_{\theta}(s_t, \epsilon_t)} \right] \\
&(\because \text{Equation A.1}) \\
&= \mathbb{E}_{s_0, \epsilon_0, \dots, \sim q} \left[\sum_{t=0}^{\infty} \sum_{k=t}^{\infty} -\gamma^k \frac{\partial g_{\theta}(s_t, \epsilon_t)}{\partial \theta} \cdot \frac{\partial r(s_k, g_{\theta}(s_t, \epsilon_t))}{\partial g_{\theta}(s_t, \epsilon_t)} \right],
\end{aligned}$$

which equals to RP gradient in Appendix A.2.2, but with different sign. However, they turn out to be the same because we minimize $L(\theta)$, but maximize $\eta(\pi_{\theta})$. Therefore, we can say that we gain RP gradient as the first gradient when we minimize Equation 2.10 for particular advantage function $\hat{A}$ and $\alpha = 1$.

A.1.6 Proof of Proposition 2.4.5

By Definition 2.4.1 and Lemma 2.4.4,

$$\begin{aligned}
\frac{\pi_{\alpha}(s, \tilde{\alpha})}{\pi_{\bar{\theta}}(s, a)} &= \frac{1}{|\det(I + \alpha \cdot \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))|} \\
&= |\det(\frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon})| \cdot |\det(\frac{\partial g_{\alpha}(s, \epsilon)}{\partial \epsilon})|^{-1},
\end{aligned}$$

where $a = g_\theta(s, \epsilon)$, and thus $\tilde{a} = g_\alpha(s, \epsilon)$. Since $\pi_{\bar{\theta}} \triangleq g_{\bar{\theta}}$, following holds:

$$\begin{aligned}
\pi_\alpha(s, \tilde{a}) &= \pi_{\bar{\theta}}(s, a) \cdot \left| \det\left(\frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon}\right) \right| \cdot \left| \det\left(\frac{\partial g_\alpha(s, \epsilon)}{\partial \epsilon}\right) \right|^{-1} \\
&= q(\epsilon) \cdot \left| \det\left(\frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon}\right) \right|^{-1} \cdot \left| \det\left(\frac{\partial g_{\bar{\theta}}(s, \epsilon)}{\partial \epsilon}\right) \right| \cdot \left| \det\left(\frac{\partial g_\alpha(s, \epsilon)}{\partial \epsilon}\right) \right|^{-1} \\
&(\because \text{Lemma 2.3.2}) \\
&= q(\epsilon) \cdot \left| \det\left(\frac{\partial g_\alpha(s, \epsilon)}{\partial \epsilon}\right) \right|^{-1},
\end{aligned}$$

which implies $\pi_\alpha \triangleq g_\alpha$.

A.2 Formulations

A.2.1 Analytical Gradient of Generalized Advantage Estimator (GAE)

GAE [schulman2015high](#) has been widely used in many RL implementations [schulman2017proximal](#), [stable-baselines3](#), [rl-games2022](#) to estimate advantages. GAE finds a balance between variance and bias of the advantage estimation by computing the exponentially-weighted average of the TD residual terms (δ_t^V) [\[28\]](#), which are defined as follows:

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t).$$

Then, GAE can be formulated as follows:

$$A_t^{GAE}(\gamma, \lambda) = \sum_{k=0}^{\infty} (\gamma \lambda)^k \delta_{t+k}^V. \quad (\text{A.2})$$

We can compute the gradients for these terms as:

$$\begin{aligned}
\frac{\partial A_t^{GAE}}{\partial a_t} &= \sum_{k=0}^{\infty} (\gamma\lambda)^k \frac{\partial \delta_{t+k}^V}{\partial a_t} \\
&= \left(\frac{1}{\gamma\lambda}\right)^t \sum_{k=0}^{\infty} (\gamma\lambda)^{t+k} \frac{\partial \delta_{t+k}^V}{\partial a_t} \\
&= \left(\frac{1}{\gamma\lambda}\right)^t \left[\sum_{k=0}^{\infty} (\gamma\lambda)^{t+k} \frac{\partial \delta_{t+k}^V}{\partial a_t} + \sum_{k=0}^{t-1} (\gamma\lambda)^k \frac{\partial \delta_k^V}{\partial a_t} \right] \\
&\quad (\because \frac{\partial \delta_k^V}{\partial a_t} = 0 \text{ for } k < t) \\
&= \left(\frac{1}{\gamma\lambda}\right)^t \sum_{k=0}^{\infty} (\gamma\lambda)^k \frac{\partial \delta_k^V}{\partial a_t} \\
&= \left(\frac{1}{\gamma\lambda}\right)^t \frac{\partial A_0^{GAE}}{\partial a_t}.
\end{aligned} \tag{A.3}$$

Based on this relationship, we can compute every $\frac{\partial A_t^{GAE}}{\partial a_t}$ with only one backpropagation of A_0^{GAE} , rather than backpropagating for every A_t^{GAE} .

A.2.2 RP Gradient Formulation

To differentiate Equation 2.1 with respect to θ , we first rewrite Equation 2.1 as follows.

$$\begin{aligned}
\eta(\pi_\theta) &= \mathbb{E}_{s_0, a_0, \dots \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \\
&= \mathbb{E}_{s_0, \epsilon_0, \dots \sim q} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, g_\theta(s_t, \epsilon_t)) \right].
\end{aligned}$$

Then, we can compute RP gradient as follows.

$$\begin{aligned}
\frac{\partial \eta(\pi_\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} \mathbb{E}_{s_0, \epsilon_0, \dots \sim q} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, g_\theta(s_t, \epsilon_t)) \right] \\
&= \mathbb{E}_{s_0, \epsilon_0, \dots \sim q} \left[\frac{\partial}{\partial \theta} \sum_{t=0}^{\infty} \gamma^t r(s_t, g_\theta(s_t, \epsilon_t)) \right] \\
&= \mathbb{E}_{s_0, \epsilon_0, \dots \sim q} \left[\sum_{t=0}^{\infty} \frac{\partial g_\theta(s_t, \epsilon_t)}{\partial \theta} \sum_{k=t}^{\infty} \gamma^k \frac{\partial r(s_k, g_\theta(s_k, \epsilon_k))}{\partial g_\theta(s_t, \epsilon_t)} \right] \\
&\quad (\because \frac{\partial r(s_k, g_\theta(s_k, \epsilon_k))}{\partial g_\theta(s_t, \epsilon_t)} \neq 0 \text{ only when } k \geq t.) \\
&= \mathbb{E}_{s_0, \epsilon_0, \dots \sim q} \left[\sum_{t=0}^{\infty} \sum_{k=t}^{\infty} \gamma^k \frac{\partial g_\theta(s_t, \epsilon_t)}{\partial \theta} \frac{\partial r(s_k, g_\theta(s_k, \epsilon_k))}{\partial g_\theta(s_t, \epsilon_t)} \right] \\
&= \mathbb{E}_{s_0, \epsilon_0, \dots \sim q} \left[\sum_{t=0}^{\infty} \sum_{k=t}^{\infty} \gamma^k \frac{\partial g_\theta(s_t, \epsilon_t)}{\partial \theta} \frac{\partial r(s_k, a_k)}{\partial a_t} \right].
\end{aligned}$$

We can estimate this RP gradient using Monte Carlo sampling and terms in Equation 2.2, and use it for gradient ascent to maximize Equation 2.1.

A.2.3 Estimator for Equation 2.4

Since $\eta(\pi_{\bar{\theta}})$ does not depend on θ , we can ignore the term and rewrite our loss function in Equation 2.4 using expectation as follows:

$$\begin{aligned}
L_{\pi_{\bar{\theta}}}(\pi_\theta) &= \int_s \rho_{\pi_{\bar{\theta}}}(s) \int_a \pi_\theta(s, a) A_{\pi_{\bar{\theta}}}(s, a) \\
&= \mathbb{E}_{s \sim \rho_{\pi_{\bar{\theta}}}, a \sim \pi_\theta(s, \cdot)} \left[A_{\pi_{\bar{\theta}}}(s, a) \right].
\end{aligned}$$

However, note that we did not collect trajectories, or experience buffer, using π_θ . Therefore, we

use another importance sampling function $q(s, a)$ [35].

$$L_{\pi_{\bar{\theta}}}(\pi_{\theta}) = \mathbb{E}_{s \sim \rho_{\pi_{\bar{\theta}}}, a \sim q(s, \cdot)} \left[\frac{\pi_{\theta}(s, a)}{q(s, a)} A_{\pi_{\bar{\theta}}}(s, a) \right].$$

Then, we can estimate this value using Monte Carlo estimation as follows, where B is the experience buffer of size N , which stores state (s_i), action (a_i), and their corresponding advantage value ($A_{\pi_{\bar{\theta}}}(s_i, a_i)$) obtained by following policy $\pi_{\bar{\theta}}$:

$$L_{\pi_{\bar{\theta}}}(\pi_{\theta}) \approx \frac{1}{N} \sum_{i=1}^N \frac{\pi_{\theta}(s_i, a_i)}{\pi_{\bar{\theta}}(s_i, a_i)} A_{\pi_{\bar{\theta}}}(s_i, a_i). \quad (\text{A.4})$$

Note that we used $\pi_{\bar{\theta}}$ in the place of q , as it is the most natural importance sampling function we can use [35, 36].

A.3 Algorithm

Here we present the details of our algorithm in Section 2.4.3. To be specific, we present 1) an empirical evidence to validate our strategy to control α , 2) the exact loss function that we use in PPO update, and 3) provide the change of α during real training to help understanding.

A.3.1 Empirical evidence

In Section 2.4.3, we argued that the estimate of $\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))$ is related to the sample variance of analytical gradients, and we can get the estimate using Lemma 2.4.4. In Figure A.1, we empirically prove the validity of this argument in the differentiable physics environments used in Section 2.5.2. In the illustration, we can observe a clear relationship between the sample variance of

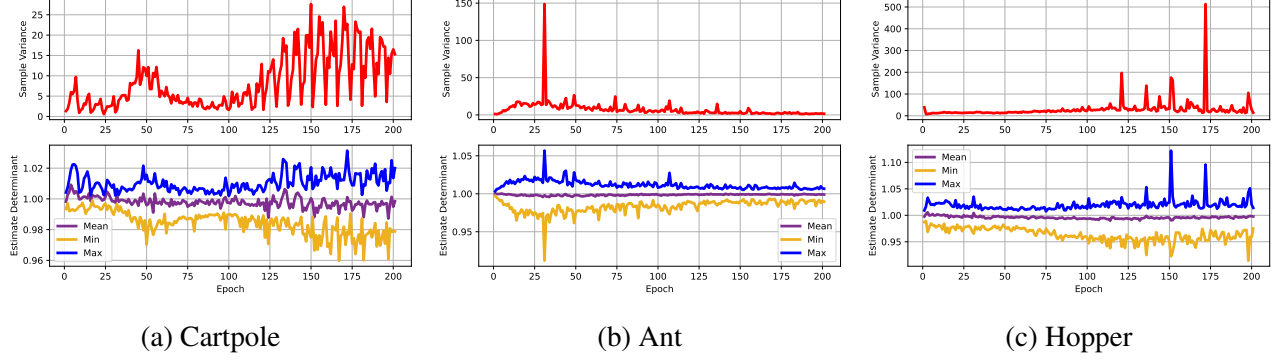


Figure A.1: The first row shows the sample variance of analytical gradients ($\frac{\partial A}{\partial a}$), and the second row shows the statistics of estimates of $\det(I + \alpha \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))$ for every state-action pair in the buffer after each epoch of training in 3 different differentiable physics environments (Cartpole, Ant, and Hopper). We used a fixed $\alpha = 10^{-2}$. Note that the minimum and maximum estimates in the second row exhibit a clear tendency to deviate from 1 more as the sample variance increases.

analytical gradients and the statistics of our estimates. Therefore, we can use these estimates as a cue to control α based on the variance of analytical gradients.

A.3.2 Loss function for PPO Update

As discussed in Section 2.3.3.2, PPO finds a better policy in the proximity of the original policy by restricting the ratio of probabilities $\pi_{\theta}(s_i, a_i)$ and $\pi_{\bar{\theta}}(s_i, a_i)$, as shown in Equation 2.5. In fact, PPO uses following surrogate loss function, which is a modified version of Equation A.4, to mandate this:

$$L_{PPO}(\theta) = \frac{1}{N} \sum_{i=1}^N \min\left(\frac{\pi_{\theta}(s_i, a_i)}{\pi_{\bar{\theta}}(s_i, a_i)} A_{\pi_{\bar{\theta}}}(s_i, a_i), \text{clip}\left(\frac{\pi_{\theta}(s_i, a_i)}{\pi_{\bar{\theta}}(s_i, a_i)}, 1 - \epsilon_{clip}, 1 + \epsilon_{clip}\right) A_{\pi_{\bar{\theta}}}(s_i, a_i)\right). \quad (\text{A.5})$$

Note that if out-of-range-ratio defined in Equation 2.12 is already 1.0, which corresponds to an extreme case, the gradient of $L_{PPO}(\theta)$ with respect to θ is 0. Therefore, PPO does not contribute to the policy update at all in this case. It justifies our decision to upper bound out-of-range-ratio by restricting α .

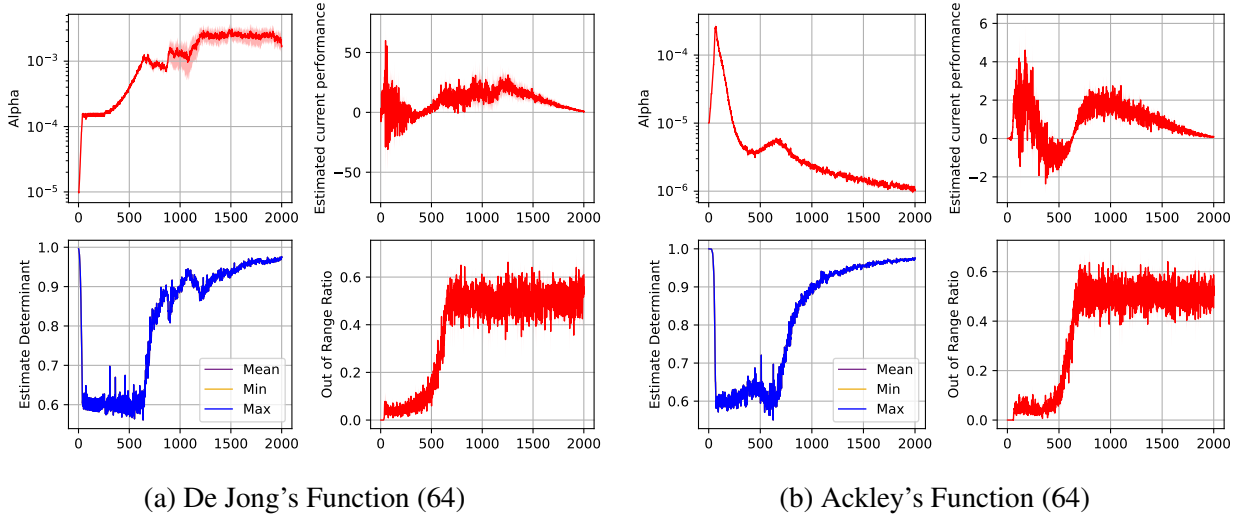


Figure A.2: Change of α during solving function optimization problems in Section 2.5.1. Since these problems are one-step problems, and observations are all the same, there is no difference between mean, min, and max of estimated determinant.

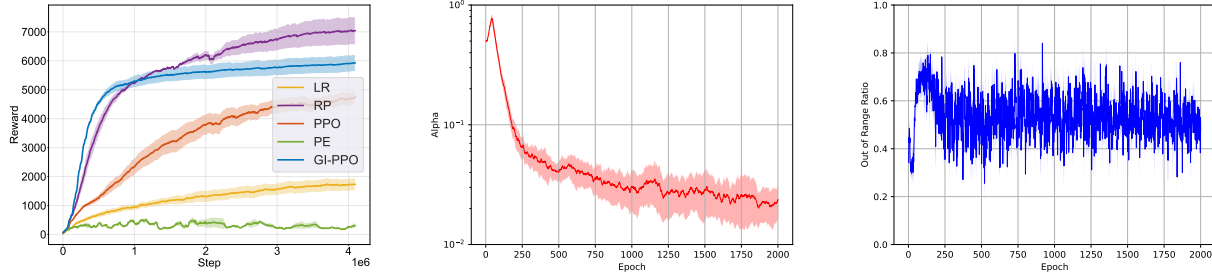
Also note that we proposed to use a virtual policy π_h instead of $\pi_{\bar{\theta}}$ in Section 2.4.3.2 to restrict PPO update to be done around π_α . This changes the importance sampling function in Equation A.5 as follows, which corresponds to our final surrogate loss function to use in PPO update:

$$L_{GIPPO}(\theta) = \frac{1}{N} \sum_{i=1}^N \min\left(\frac{\pi_\theta(s_i, a_i)}{\pi_h(s_i, a_i)} A_{\pi_{\bar{\theta}}}(s_i, a_i), \text{clip}\left(\frac{\pi_\theta(s_i, a_i)}{\pi_h(s_i, a_i)}, 1 - \epsilon_{clip}, 1 + \epsilon_{clip}\right) A_{\pi_{\bar{\theta}}}(s_i, a_i)\right). \quad (\text{A.6})$$

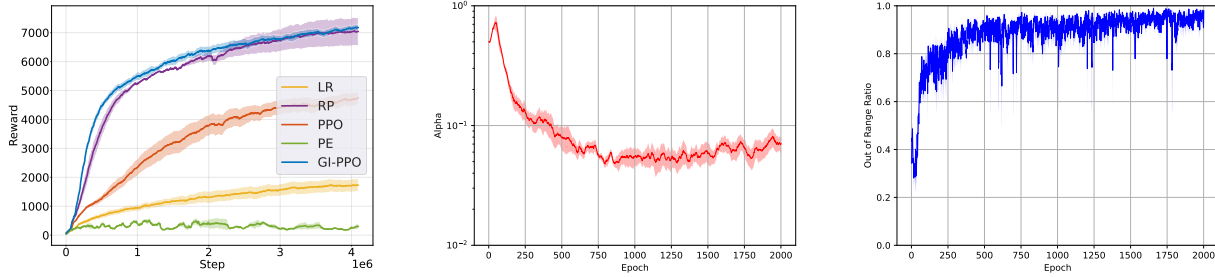
A.3.3 Example: Function Optimization Problems (Section 2.5.1)

Here we present how α changes as we solve function optimization problems in Section 2.5.1 with Figure A.2. There are 4 plots for each of the problems.

- Alpha: Shows change of α over training epoch.
- Estimated current performance: Shows expected additional return, which corresponds to Equa-



(a) Statistics of training in **Ant** environment when $\delta_{oorr} = 0.5$.



(b) Statistics of training in **Ant** environment when $\delta_{oorr} = 1.0$.

Figure A.3: Change of α during training in Ant environment in Section 2.5.2. In this environment, maximum out-of-range-ratio (δ_{oorr}) plays a major role in bounding α from growing. Observe that GI-PPO achieves better results than RP when we set $\delta_{oorr} = 1.0$.

tion A.4 (R_α in Algorithm 1).

- Estimate Determinant: Shows statistics of estimated $\det(I + \alpha \cdot \nabla_a^2 A_{\pi_{\bar{\theta}}}(s, a))$.
- Out of Range Ratio: Shows out-of-range-ratio in Equation 2.12.

For these problems, we have set $\alpha_0 = 10^{-5}$, $\delta_{det} = 0.4$, and $\delta_{oorr} = 0.5$.

For De Jong’s function (Figure A.2a), we can observe that α shows steady increase over time, but it is mainly upper bounded by variance criterion and out-of-range-ratio. In the end, α stabilizes around 10^{-3} . In contrast, for Ackley’s function (Figure A.2b) we can see that α increases rapidly at first, but it soon decreases due to the variance and bias criteria. Compare these graphs with optimization curve in Figure 2.2d. Then we would be able to observe that this large α contributes to the faster convergence of GI-PPO compared to PPO at the early stage. However, after that, α decreases slowly due to the

out-of-range-ratio. In the end, α reaches approximately 10^{-6} , which is much lower than 10^{-3} of De Jong’s function. These observations align well with our intuition that RP gradient would play bigger role for De Jong’s function than Ackley’s function, because De Jong’s function exhibits RP gradients with much lower variance than those of Ackley’s.

A.3.4 Example: Physics Simulation Problems (Section 2.5.2)

In Section 2.5.2, GI-PPO could not achieve better results than RP in Ant and Hopper environments. For Ant environment, we found out that the hyperparameter δ_{oorr} was a bottleneck that bounds GI-PPO’s performance. In Figure A.3, we illustrated the change of α during training along the change of out-of-range-ratio. When we set $\delta_{oorr} = 0.5$, α keeps decreasing over time to keep the ratio around 0.5, which means the diminishing role of RP gradients in training. In contrast, when we set $\delta_{oorr} = 1.0$, after 750 epochs, α does not decrease anymore, and out-of-range ratio stays near 1. This means that PPO rarely affects the training – policy update is almost driven by RP gradients.

Likewise, there are environments where RP gradients are much more effective than the other policy gradients. In those situations, it would be more desirable to solely depend on RP gradients without considering PPO. Since our method considers RP gradients in the PPO viewpoint, it is hard to detect such cases yet.

A.4 Experimental Details

A.4.1 Baseline Methods

Here we explain implementation details of the baseline methods that were used for comparisons in Section 2.5. First, we’d like to point out that we used relatively a short time horizon to collect

experience, and used the critic network for bootstrapping, instead of collecting experience for the whole trajectory for all the methods. Therefore, the collected experience could be biased. However, we chose to implement in this way, because it allows faster learning than collecting the whole trajectory in terms of number of simulation steps.

A.4.1.1 LR

We can estimate original LR gradient as follows, using log-derivative trick [29, 30]

$$\begin{aligned}\frac{\partial \eta(\pi_\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} \mathbb{E}_{s_0, a_0, \dots \sim \pi_\theta} \left[L(s_0, a_0, \dots) \sum_{t=0}^{\infty} \log \pi_\theta(s_0, a_0) \right] \\ &= \frac{\partial}{\partial \theta} \mathbb{E}_{s_0, a_0, \dots \sim \pi_\theta} \left[\sum_{t=0}^{\infty} L(s_0, a_0, \dots) \log \pi_\theta(s_0, a_0) \right],\end{aligned}$$

where $L(s_0, a_0, \dots) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$ is the discounted accumulated return. However, this LR gradient often suffers from high variance, because L varies a lot from trajectory to trajectory. To reduce such variance and faithfully compare our method to LR gradient-based method, we decided to use advantage function in the place of L , and particularly used GAE [34] as the advantage function as we used for PPO. After calculating this LR gradient, we take a gradient ascent step with pre-defined learning rate.

A.4.1.2 RP

By using RP gradient in Appendix A.2.2, we can perform gradient ascent as we did in LR. Also, because we use short time horizon and critic network, this method is very similar to SHAC [14]. Please refer to the paper for more details.

A.4.1.3 PPO

Our PPO implementation is based on that of RL Games [310]. However, to be as fair as possible, we used the same critic network as the other methods, instead of using that implemented in the RL library. Also, we omitted several other additional losses that are not vital to PPO’s formulation, such as entropy loss.

A.4.1.4 LR+RP

After we compute LR (∇_{LR}) and RP (∇_{RP}) gradient as shown above, we can interpolate them using their sample variance as follows [50].

$$\begin{aligned}\nabla_{LR+RP} &= \nabla_{LR} \cdot \kappa_{LR} + \nabla_{RP} \cdot (1 - \kappa_{LR}), \\ \kappa_{LR} &= \frac{\sigma_{RP}^2}{\sigma_{RP}^2 + \sigma_{LR}^2},\end{aligned}$$

where σ_{RP} and σ_{LR} are sample standard deviation of RP and LR gradients. We gain these terms by computing trace of covariance matrix of the sample gradients from different trajectories.

Since we have to compute this sample statistics, we have to do multiple different backpropagations for different trajectories, which incur a lot of computation time. Also, we found out that computing covariance matrix is also time consuming when controller has a large number of parameters. Therefore, we decided to use only limited number of sample gradients (16) to compute sample variance, and also truncate the gradient to smaller length (512) to facilitate computation.

A.4.1.5 PE

We tried to faithfully re-implement policy enhancement scheme of [53].

A.4.2 Network architecture and Hyperparameters

In this section, we provide network architectures and hyperparameters that we used for experiments in Section 2.5. For each of the experiments, we used the same network architectures, the same length of time horizons before policy update, and the same optimization procedure for critic updates, etc. We present these common settings first for each of the problems.

For GI-PPO, there are hyperparameters for update towards α -policy, and those for PPO update. We denote the hyperparameters for α -policy update by appending (α), and those for PPO update by appending (PPO). For the definition of hyperparameters for α -policy update, please see Algorithm 1 for details.

A.4.2.1 Function Optimization Problems (Section 2.5.1)

For these problems, common settings are as follows.

- Actor Network: MLP with [32, 32] layers and ELU activation function
- Critic Network: MLP with [32, 32] layers and ELU activation function
- Critic Hyperparameters: Learning rate = 10^{-3} , Iterations = 16, Batch Size = 4
- Number of parallel environments: 64
- Horizon Length: 1

- γ (Discount factor): 0.99
- τ (GAE): 0.95

Hyperparameters for **LR** are as follows.

- Learning Rate: Dejong(1), Dejong(64) = 10^{-3} , Ackley(1) = 10^{-4} , Ackley(64) = $3 \cdot 10^{-4}$
- Learning Rate Scheduler: Linear¹

Hyperparameters for **RP** are as follows.

- Learning Rate: Dejong(1), Dejong(64) = 10^{-2} , Ackley(1), Ackley(64) = 10^{-3}
- Learning Rate Scheduler: Linear

Hyperparameters for **LR+RP** are as follows.

- Learning Rate: Dejong(1), Dejong(64) = 10^{-3} , Ackley(1) = 10^{-4} , Ackley(64) = $3 \cdot 10^{-4}$
- Learning Rate Scheduler: Linear

Hyperparameters for **PPO** are as follows.

- Learning Rate: Dejong(1), Ackley(1) = 10^{-4} , Dejong(64), Ackley(64) = 10^{-2}
- Learning Rate Scheduler: Constant
- Batch Size for Actor Update: 64
- Number of Epochs for Actor Update: 5

¹Learning rate decreases linearly to the minimum value as learning progresses.

- ϵ_{clip} : 0.2

Hyperparameters for **GI-PPO** are as follows.

- (α) Learning Rate: 10^{-3}
- (α) Batch Size for Actor Update: 64
- (α) Number of Epochs for Actor Update: 16
- (α) α_0 : 10^{-5}
- (α) $\max(\alpha)$: 1.0
- (α) β : 1.1
- (α) δ_{det} : 0.4
- (α) δ_{orr} : 0.5
- (PPO) Learning Rate: Dejong(1), Ackley(1) = 10^{-4} , Dejong(64), Ackley(64) = 10^{-2}
- (PPO) Learning Rate Scheduler: Constant
- (PPO) Batch Size for Actor Update: 64
- (PPO) Number of Epochs for Actor Update: 5
- (PPO) ϵ_{clip} : 0.2

A.4.2.2 Differentiable Physics Problems (Section 2.5.2)

For these problems, common settings are as follows.

- Actor Network: MLP with [64, 64] (Cartpole), and [128, 64, 32] (Ant, Hopper) layers and ELU activation function
- Critic Network: MLP with [64, 64] layers and ELU activation function
- Critic Hyperparameters: Learning rate = 10^{-3} (Cartpole), $2 \cdot 10^{-3}$ (Ant), and $2 \cdot 10^{-4}$ (Hopper), Iterations = 16, Batch Size = 4
- Number of parallel environments: 64(Cartpole, Ant), 256(Hopper)
- Horizon Length: 32
- γ (Discount factor): 0.99
- τ (GAE): 0.95

Hyperparameters for **LR** are as follows.

- Learning Rate: 10^{-4}
- Learning Rate Scheduler: Linear

Hyperparameters for **RP** are as follows.

- Learning Rate: Cartpole = 10^{-2} , Ant, Hopper = $2 \cdot 10^{-3}$
- Learning Rate Scheduler: Linear

Hyperparameters for **PPO** are as follows.

- Learning Rate: Cartpole = $3 \cdot 10^{-4}$, Ant, Hopper = 10^{-4}
- Learning Rate Scheduler: Cartpole = Adaptive², Ant, Hopper = Constant
- Batch Size for Actor Update: 2048
- Number of Epochs for Actor Update: 5
- ϵ_{clip} : 0.2

Hyperparameters for **GI-PPO** are as follows.

- (α) Learning Rate: Cartpole = 10^{-2} , Ant = $5 \cdot 10^{-4}$, Hopper = $5 \cdot 10^{-3}$
- (α) Batch Size for Actor Update: Cartpole, Ant = 2048, Hopper = 8192
- (α) Number of Epochs for Actor Update: 16
- (α) α_0 : Cartpole, Ant = $5 \cdot 10^{-1}$, Hopper = $5 \cdot 10^{-3}$
- (α) $\max(\alpha)$: 1.0
- (α) β : 1.02
- (α) δ_{det} : 0.4
- (α) δ_{orr} : Cartpole, Hopper = 0.75, Ant = 0.5
- (PPO) Learning Rate: Cartpole = $3 \cdot 10^{-4}$, Ant, Hopper = 10^{-4}

²Learning rate is adaptively controlled, so that the KL divergence between the updated policy and the original policy is maintained at certain value, 0.008 in this case.

- (PPO) Learning Rate Scheduler: Cartpole = Adaptive, Ant, Hopper = Constant
- (PPO) Batch Size for Actor Update: 2048
- (PPO) Number of Epochs for Actor Update: 5
- (PPO) ϵ_{clip} : 0.2

A.4.2.3 Traffic Problems (Section 2.5.3)

For these problems, common settings are as follows.

- Actor Network: MLP with [512, 64, 64] layers and ELU activation function
- Critic Network: MLP with [64, 64] layers and ELU activation function
- Critic Hyperparameters: Learning rate = 10^{-3} , Iterations = 16, Batch Size = 4
- Number of parallel environments: 64
- Horizon Length: 32
- γ (Discount factor): 0.99
- τ (GAE): 0.95

Hyperparameters for **LR** are as follows.

- Learning Rate: $3 \cdot 10^{-4}$
- Learning Rate Scheduler: Linear

Hyperparameters for **RP** are as follows.

- Learning Rate: 10^{-3}
- Learning Rate Scheduler: Linear

Hyperparameters for **LR+RP** are as follows.

- Learning Rate: $3 \cdot 10^{-4}$
- Learning Rate Scheduler: Linear

Hyperparameters for **PPO** are as follows.

- Learning Rate: $3 \cdot 10^{-4}$
- Learning Rate Scheduler: Constant
- Batch Size for Actor Update: 2048
- Number of Epochs for Actor Update: 5
- ϵ_{clip} : 0.2

Hyperparameters for **GI-PPO** are as follows.

- (α) Learning Rate: 10^{-5}
- (α) Batch Size for Actor Update: 2048
- (α) Number of Epochs for Actor Update: 16
- (α) α_0 : 10^{-1}
- (α) $\max(\alpha)$: 1.0

- $(\alpha) \beta$: 1.1
- $(\alpha) \delta_{det}$: 0.4
- $(\alpha) \delta_{orr}$: 0.5
- (PPO) Learning Rate: $3 \cdot 10^{-4}$
- (PPO) Learning Rate Scheduler: Constant
- (PPO) Batch Size for Actor Update: 2048
- (PPO) Number of Epochs for Actor Update: 5
- (PPO) ϵ_{clip} : 0.2

A.5 Problem Definitions

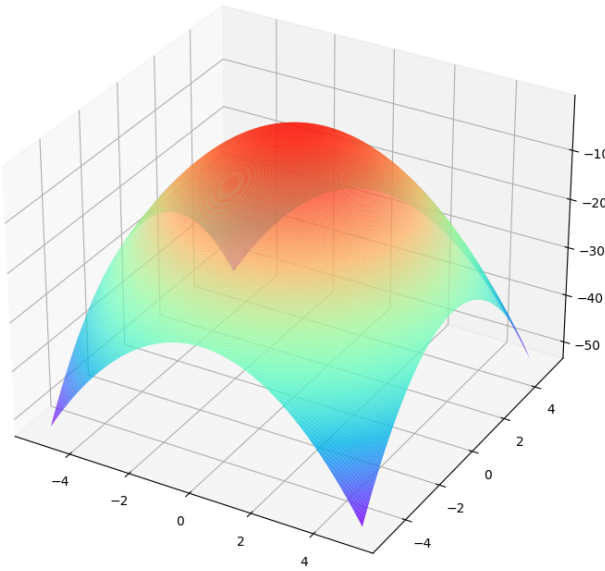
Here we present details about the problems we suggested in Section 2.5.

A.5.1 Function Optimization Problems (Section 2.5.1)

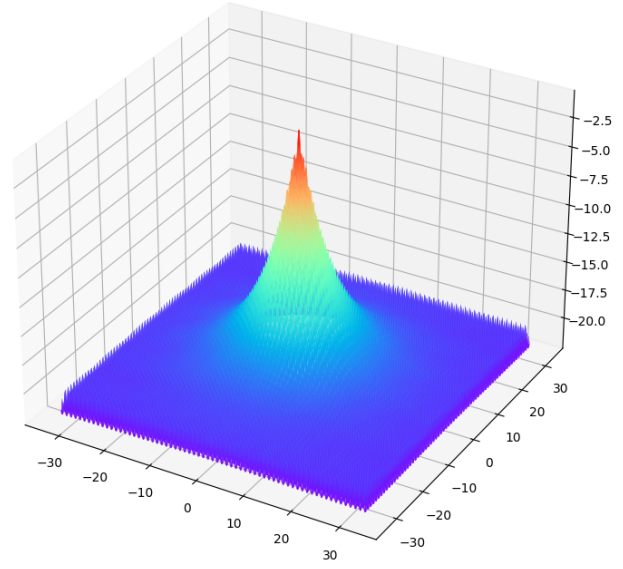
(N-dimensional) De Jong's function (F_D) and Ackley's function (F_A) are defined for n -dimensional vector x and are formulated as follows [172]:

$$F_D(x) = \sum_{i=1}^n x_i^2,$$

$$F_A(x) = -20 \cdot \exp(-0.2 \cdot \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}) - \exp(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)) + 20 + \exp(1).$$



(a) De Jong's Function



(b) Ackley's Function

Figure A.4: Landscape of target functions in 2 dimensions.

As we mentioned in Section 2.5.1, we multiply -1 to these functions to make these problems maximization problems. Also, even though these two functions have their optimum at $x = 0$, they exhibit very different landscape as shown in Figure A.4.

When it comes to the formal definition as RL environments, we can define these problems as follows.

- Episode Length: 1
- Observation: $[0]$
- Action: n -dimensional vector $\mathbf{x}$, all of which element is in $[-1, 1]$.
- Reward: De Jong's Function = $F_D(5.12\mathbf{x})$, Ackley's Function = $F_A(32.768\mathbf{x})$

A.5.2 Traffic Problems (Section 2.5.3)

Before introducing pace car problem, which we specifically discussed in this paper, we'd like to briefly point out the traffic model that we used to simulate motions of individual vehicles in the traffic environment.

A.5.2.1 Traffic Model

In our traffic simulation, the state $s \in \mathbb{R}^{2N}$ is defined as a concatenation of all vehicle states, $q_n \in \mathbb{R}^2$, where $1 \leq n \leq N$ stands for vehicle ID. q_n can be represented simply with the vehicle's position (x_n) and velocity (v_n).

$$q_n = \begin{bmatrix} x_n \\ v_n \end{bmatrix}$$

$$s = \begin{bmatrix} q_1^T, \dots, q_N^T \end{bmatrix}^T,$$

where n is the vehicle ID. Since this traffic state s changes over time, we represent the traffic state at timestep t as s_t . Then our differentiable traffic simulator is capable of providing the gradient of the simulation state at the next timestep $t + 1$ with respect to the state at the current timestep t , or $\frac{ds_{t+1}}{ds_t}$.

This is possible because we use the Intelligent Driver Model (IDM) to model car-following behavior in our simulator [311], which is differentiable. IDM describes vehicle speed $\dot{x}$ and acceleration $\dot{v}$ as a function of desired velocity v_0 , safe time headway in meters T , maximum acceleration a ,

comfortable deceleration b , the minimum distance between vehicles in meters δ , vehicle length l , and difference in speed with the vehicle in front Δv_α as follows:

$$\begin{aligned}\dot{x}_\alpha &= \frac{dx_\alpha}{dt} = v_\alpha, \\ \dot{v}_\alpha &= \frac{dv_\alpha}{dt} = a \left(1 - \left(\frac{v_\alpha}{v_0} \right)^\delta - \left(\frac{s^*(v_\alpha, \Delta v_\alpha)}{s_\alpha} \right)^2 \right), \\ s_\alpha &= x_{\alpha-1} - x_\alpha - l_{\alpha-1}, \\ s^*(v_\alpha, \Delta v_\alpha) &= s_0 + v_\alpha T + \frac{v_\alpha \Delta v_\alpha}{2\sqrt{ab}}, \\ \Delta v_\alpha &= v_\alpha - v_{\alpha-1},\end{aligned}$$

where α means the order of the vehicle in the lane. Therefore, $(\alpha - 1)$ -th vehicle runs right in front of α -th vehicle, and this relationship plays an important role in IDM. Also note that the computed acceleration term ($\dot{v}$) is differentiable with respect to the other IDM variables.

IDM is just one of many car-following models used in traffic simulation literature. We choose IDM for its prevalence in previous mixed autonomy literature, however, any ODE-based car-following model will also work in our simulator as far as it is differentiable. In our simulator, automatic differentiation governs the gradient computation.

A.5.2.2 Lane

Under IDM, lane membership is one of the most vital variables for simulation. This is because IDM builds on the leader-follower relationship. When a vehicle changes its lane with lateral movement, as shown as red arrows in Figure A.5, such relationships could change, and our gradients do not convey any information about it, because lane membership is a discrete variable in nature. Note that

this lateral movement also affects the longitudinal movement, rendered in green arrows in Figure A.5, of a vehicle, and gradient from it is valid as far as the lane membership does not change. However, when the lane membership changes, even if our simulator gives gradient, it tells us only “partial” information in the sense that it only gives information about longitudinal behavior. Therefore, we could say that analytical gradients we get from this environment is “incomplete” (not useless at all, because it still gives us information about longitudinal movement), and thus biased.

A.5.2.3 Pace Car Problem

In this problem, there is a single autonomous vehicle that we have to control to regulate other human driven vehicles, which follow IDM, to run at the given target speed. Since human driven vehicles control their speeds based on the relationship to their leading vehicles, our autonomous vehicle has to control itself to run in front of those human driven vehicles and adjust their speeds to the target speed. The number of lanes and human driven vehicles varies across different environments as follows.

- Single Lane: Number of lanes = 1, Number of vehicles per lane = 1
- 2 Lanes: Number of lanes = 2, Number of vehicles per lane = 2
- 4 Lanes: Number of lanes = 4, Number of vehicles per lane = 4
- 10 Lanes: Number of lanes = 10, Number of vehicles per lane = 1

See Figure A.6 for 10-lane environment.

Then we can formally define RL environments as follows, where N is the number of human driven vehicles in total, and v_{tgt} is the target speed.

- Episode Length: 1000

- Observation: $s \in \mathbb{R}^{2N}$
- Action: $\mathbf{a} \in \mathbb{R}^2$ ($\mathbf{a}_0$ = Acceleration, $\mathbf{a}_1$ = Steering)
- Reward: $1 - \frac{1}{N} \sum_{i=1}^N \min(\frac{|v_i - v_{tgt}|}{v_{tgt}}, 1)$

However, to facilitate learning, we additionally adopted following termination conditions to finish unpromising trials early.

- Terminate when autonomous vehicle collides with human driven vehicle
- Terminate when autonomous vehicle goes out of lane
- Terminate when autonomous vehicle runs too far away from the other vehicles
- Terminate when autonomous vehicle falls behind the other vehicles

When one of these conditions is met, it gets reward of -1 and the trial terminates.

A.6 Renderings

A.6.1 Traffic Simulation

A.6.1.1 Gradient flow

As described in Appendix [A.5.2](#), analytical gradients are only valid along longitudinal direction, as illustrated with green arrows in Figure [A.5](#). When a vehicle changes lane with lateral movement, as shown with red arrows, even though such change would incur different behaviors of following vehicles in multiple lanes, analytical gradients cannot convey such information. However, they still

tell us information about longitudinal behavior - which is one of the motivations of our research, to use analytical gradients even in these biased environments.

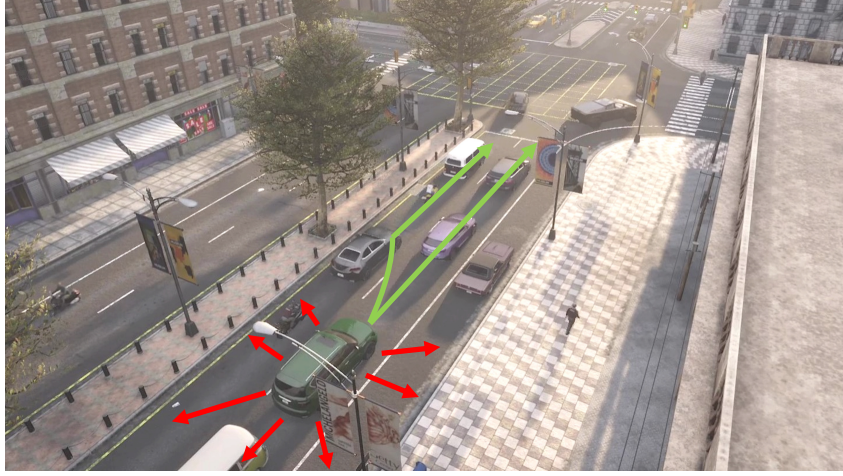


Figure A.5: **Traffic Environment.** In traffic flows, only partial gradients of forward simulation are known; acceleration along a (longitudinal) traffic lane is continuous and admits gradient flow (green), while (lateral) lane-changing is a discrete event and thus prohibits gradient flow (red). Analytical gradients convey only partial information about traffic dynamics.

A.6.1.2 Pace Car Problem

We additionally provide renderings of the 10-lane pace car problem defined in Appendix A.5.2. In Figure A.6, we can observe 10 parallel lanes, 10 human driven vehicles rendered in yellow, and the autonomous vehicle rendered in blue that we have to control. As shown there, human driven vehicles adjust their speeds based on IDM when the autonomous vehicle blocks their way. Therefore, the autonomous vehicle has to learn how to change their lanes, and also how to run in appropriate speed to regulate the following vehicles. Our experimental results in Section 2.5.3 show that our method achieves better score than the baseline methods by adopting biased analytical gradients in PPO.

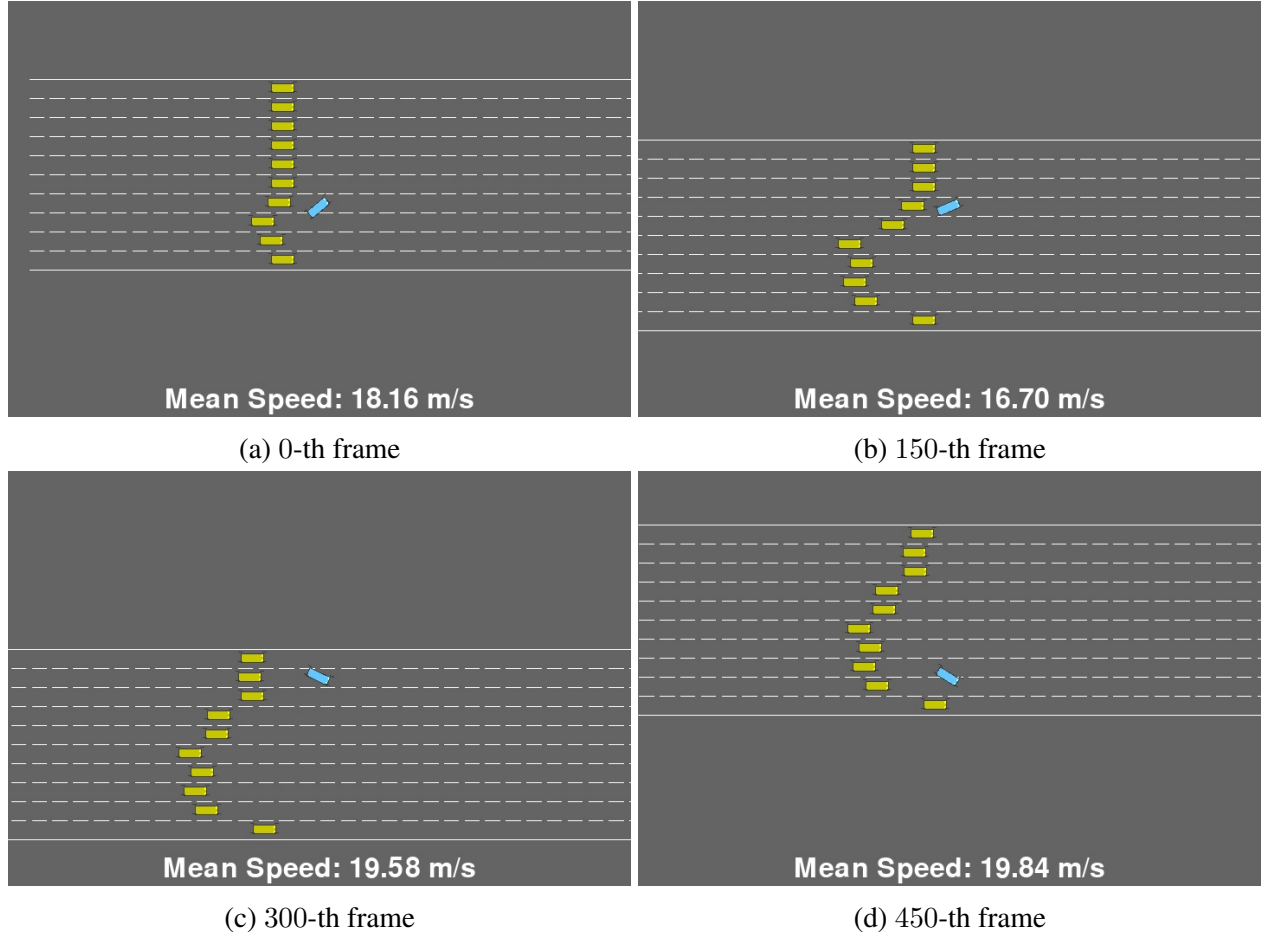


Figure A.6: 10-Lane pace car environment. The autonomous vehicle (Blue) has to interfere paths of the other human driven vehicles (Yellow) to limit their speeds to $10m/s$. Even though it is a hard problem to achieve high score, our method achieved far better score than the baseline methods.

A.7 Computational Costs

We provide wall clock training times for each method in traffic environments to finish all the training epochs in Table A.1. All of the trainings were done based on the settings in Appendix A.5.2. Note that the longer training time does not always mean that the method is worse than the others – the method could have achieved better results in shorter time, while requiring much longer time to complete all the training epochs.

In Table A.1, we can observe that GI-PPO requires a little more time than RP, as it computes analytical gradients as RP does, but requires an additional time for controlling α and doing PPO update. However, the additional cost is not so significant. In contrast, LR+RP consumes much more time than the other methods, even though it is based on the same spirit as ours. This is because it has to backpropagate through each trajectory in the experience buffer to estimate the sample variance of the gradients. Therefore, we can see that our method attains better results than LR+RP (Figure 2.4) with lower computational cost.

Table A.1: Average (wall clock) training time (sec) for traffic problems.

Problem	LR	RP	PPO	LR+RP	GI-PPO
Single Lane	120	282	181	1335	332
2 Lanes	142	330	218	1513	411
4 Lanes	304	646	366	2093	813
10 Lanes	244	496	294	2103	533

Appendix B: Time-Aware World Model for Adaptive Prediction and Control

B.1 Descriptions of 4th-order Runge-Kutta integration.

In this section, we extend our description of the 4th-order Runge-Kutta (RK4) integration mentioned in Section 3.4.1.2. The detailed RK4 integration is as follows:

$$\begin{aligned}k_1 &= d(z_t, a_t, \Delta t) \\ \hat{z}_1 &= z_t + d(z_t, a_t, \Delta t/2) \cdot \tau(\Delta t/2) \\ k_2 &= d(\hat{z}_1, a_t, \Delta t) \\ \hat{z}_2 &= z_t + d(\hat{z}_1, a_t, \Delta t/2) \cdot \tau(\Delta t/2) \\ k_3 &= d(\hat{z}_2, a_t, \Delta t) \\ \hat{z}_3 &= z_t + d(\hat{z}_2, a_t, \Delta t) \cdot \tau(\Delta t) \\ k_4 &= d(\hat{z}_3, a_t, \Delta t) \\ \hat{z}_{t+\Delta t} &= z_t + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \cdot \tau(\Delta t)\end{aligned}\tag{B.1}$$

Consistent with the notations in Section 3.4.1.2, z_t, a_t denotes the latent state-action pairs at time t , $d(\cdot)$ denotes our dynamic model parameterized by a neural network, and $\hat{z}_i$ ($i \in 1, 2, 3$) are the intermediate middle points. The final prediction of next latent state under time step size Δt is $\hat{z}_{t+\Delta t}$.

B.2 Meta-World Task Dynamics Visualizations.

To illustrate the diversity and key differences in dynamics across control tasks in our experiments, we present additional visualizations of different Meta-World tasks. Figure B.2 and Figure B.3 show sequences of renderings from the task initialization to the task completion (from left to right) of different control tasks.

B.3 Additional results for ablation study of Δt sampling strategies.

In this section, we provide additional results for the ablation study on the impact of Δt sampling strategy on the TAWM’s performance and efficiency. Specifically, we trained the same TAWM over the same range of Δt values using two different sampling strategies: **(1)** Log-Uniform(1 ms, 50 ms) and **(2)** Uniform(1 ms, 50 ms). The performance of TAWMs trained with different Δt sampling strategies, along with the non-time-aware baselines, is shown in Figure B.1.

Figure B.1 shows that while our time-aware models trained with the uniform sampling strategy generally perform better across most environments – and significantly outperform at low sampling

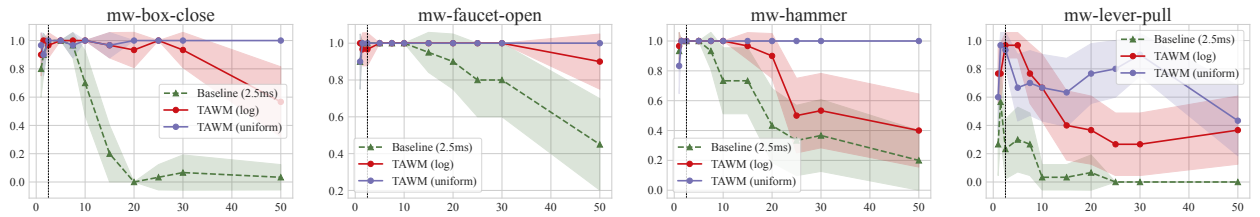


Figure B.1: Ablation Study on Δt Sampling Strategy. We trained our TAWM models with **log-uniform** and **uniform** sampling strategies, both of which use the RK4 integration method. The **baseline** method was trained using only default time step ($\Delta t = 2.5ms$), which is signified with the black dashed lines. The x-axis corresponds to the evaluation Δt , and the y-axis corresponds to the success rate. Means and 95% confidence intervals are computed over 3 seeds, each evaluated for 10 episodes.



Figure B.2: **Visualizations of the Meta-World control tasks.** From top to bottom: (1) *Assembly*, (2) *Basketball*, (3) *Box Close*, (4) *Faucet Open*, (5) *Hammer*. From left to right: sequences of renderings from task initialization to completion.

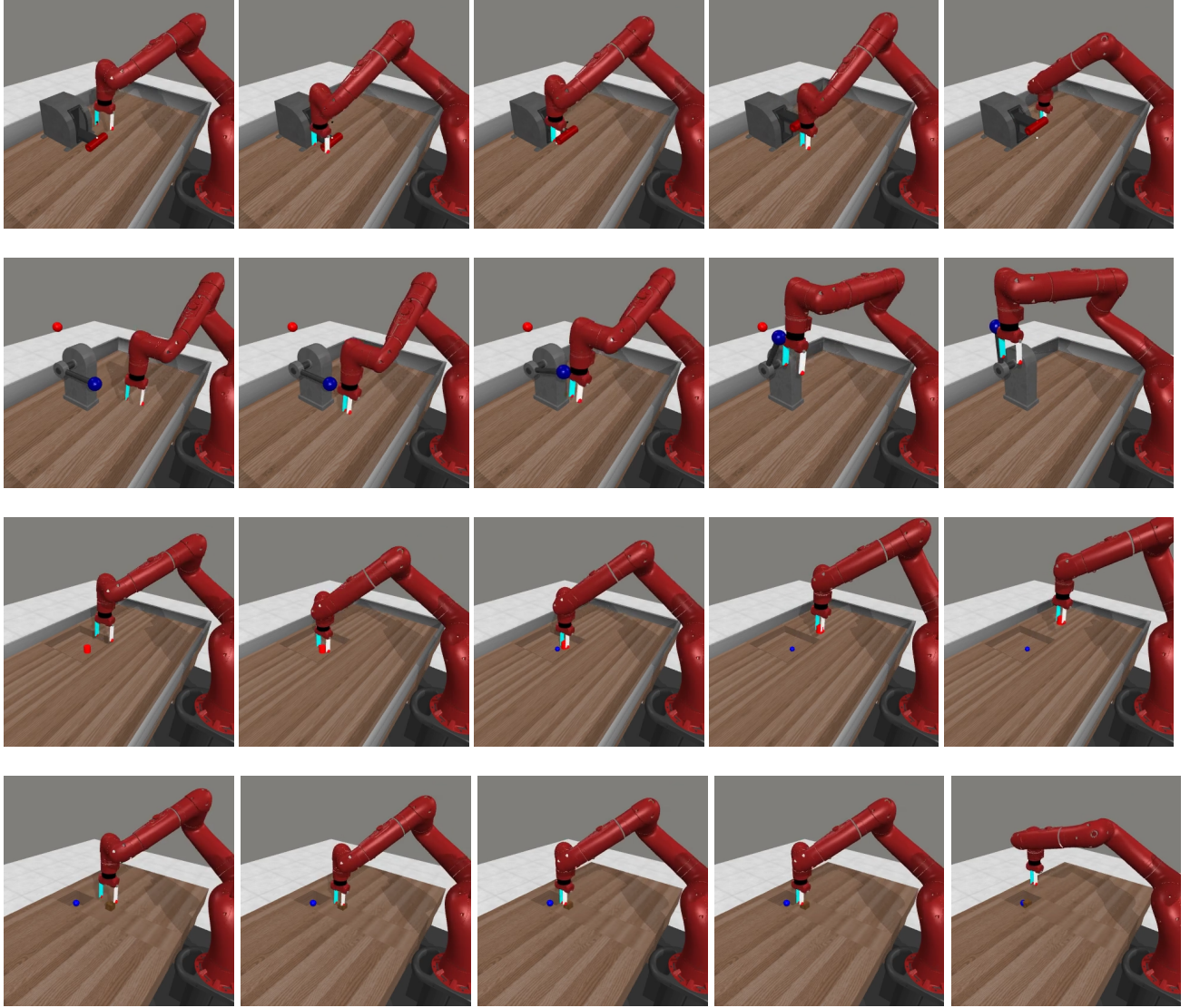


Figure B.3: **Visualizations of the Meta-World control tasks.** From top to bottom: (6) *Handle Pull*, (7) *Lever Pull*, (8) *Pick Out Of Hole*, (9) *Sweep Into*. From left to right: sequences of renderings from task initialization to completion.

rates (inference $\Delta t \geq 30$ ms) – they exhibit lower success rates at smaller inference Δt in certain environments, such as Meta-World Lever-Pull. Regardless of the Δt sampling strategy, TAWMs consistently outperform the non-time-aware baselines. Therefore, **our time-aware model can be efficiently and effectively trained with any reasonable sampling strategy and is not restricted to either log-uniform or uniform sampling.**

Developing a more adaptive and optimal Δt sampling strategy tailored to specific objectives and scenarios is a promising direction for future work – particularly to achieve strong performance across both small and large Δt values and to accelerate convergence rates. In the meantime, TAWM with either log-uniform or uniform sampling strategy has demonstrated practical effectiveness, as shown by our experimental results.

B.4 Comparisons with Multi Time Scale World Model (MTS3).

As mentioned in Section 3.2, Multi Time Scale World Model (MTS3) [192] is a closely related work sharing the high-level motivation with our work: to model the world dynamics at multiple temporal levels. Specifically, MTS3 proposes a probabilistic approach to jointly learn the world dynamics at two temporal abstractions: task level (slow dynamics/timescale) and state level (fast dynamics/timescale). These 2 timescales are separately learned by two state space models (SSMs): SSM^{fast} and SSM^{slow} , where SSM^{fast} learns the dynamics evolving at original small timestep Δt of the dynamical systems and SSM^{slow} learn the slow dynamics evolving at $H\Delta t$. Although this approach also explicitly considered different temporal abstraction levels in learning the world dynamics, there are several critical differences between MTS3 compared to our work:

1. **Models vs Training method:** [192] proposes a model architecture to learn a world model with

several discrete temporal abstraction levels. On the other hand, we proposed a *simple yet effective and efficient time-aware training method* that can be employed to train any world model architecture.

2. **Discrete vs continous timescales:** The original MTS3 currently only handle only 2 timescales: Δt and $H\Delta t$, where both Δt and H is fixed in the training process. Although the MTS3 can be adapted to learn multiple timescales, the number of timescales is limited to a discrete value. Furthermore, the SSM^{slow} (slow dynamic model) does not directly model state transition under large temporal gap Δt (or low observation rate) but rather learns the task latents to guide SSM^{fast} to long-horizon predictions. On the other hand, our time-aware approach can directly predict the future states $s_{t+\Delta t}$ under large Δt .
3. **Multi-step vs one-step prediction:** MTS3 considers the future prediction under large Δt as a long-horizon prediction problem. Specifically, to predict $s_{t+\Delta t}$, MTS3 discretizes the long temporal gap into several smaller timesteps: $\Delta t = M\Delta t_{fast}$, where Δt_{fast} is the original timestep size SSM^{fast} is trained with and $M \in \mathbb{N}^+$. MTS3 then iteratively applies the model M times to predict $s_{t+\Delta t}$. This timestep discretization approach has 2 critical limitations: **(1)** MTS3 cannot model state transitions under Δt that is not divisible by Δt_{fast} and **(2)** multi-step predictions are vulnerable to compounding errors, a well-known problem in long-horizon modeling. On the other hand, our model can directly predict the next state with a *one-step prediction*, effectively alleviating the compounding error problem.
4. **Inference efficiency:** Another disadvantage of multi-step prediction is inference inefficiency. In contrast, our time-aware model can efficiently predict long-term future states without sacrificing computational efficiency by using one-step prediction.

5. Prediction-only vs Control: As acknowledged by [192], the original MTS3 is strictly a prediction model. On the contrary, our time-aware model can be used efficiently with a planner to solve control problems.

We conduct empirical comparisons between MTS3 and our proposed time-aware model on the control problem, extending beyond the prediction-only scope in MTS3. First, MTS3 is trained with offline data consisting of 4×10^6 (4M transitions) collected from random trajectories (10%), partially-trained policy’s trajectories (20%), and fully-trained expert policy trajectories (80%). Since MTS3 is strictly prediction-focused and is not designed for controls, we combined MTS3 with MPPI planners and our world model’s trained value and reward functions. Implementation-wise, we replaced our dynamic model with MTS3 and kept all other components unchanged, including the planner (MPPI), learned value function, and learned reward function. This design ensures a fair comparison between the models, as any performance gap is attributed solely to the difference between MTS3 and our dynamic model.

We kept the default hyperparameter settings as in the original MTS3 paper and codebase while varying the slow dynamics H to investigate the impacts of H on MTS3’s performance. The authors of MTS3 suggested that using $H = \sqrt{T}$, which is $\sqrt{99} \approx 10$ in our experiments on Meta-World environments. We chose $H = 11$ to divide the episodes into equal-length local SSM windows. The MTS3 inference stepping are adjusted such that when evaluated on $\Delta t_{eval} > \Delta t_{train}$, the model is applied $\Delta t_{eval} / \Delta t_{train}$ times (Δt_{train} is the fast time step between SSM^{fast} ’s observations).

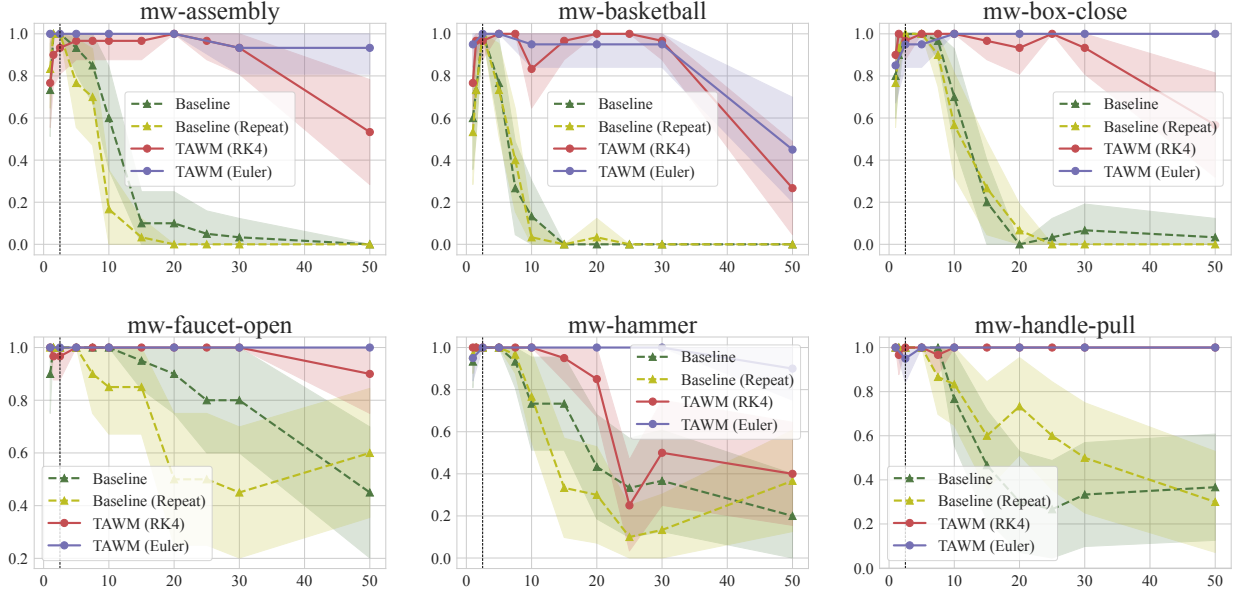


Figure B.4: **Performance comparisons for 6 additional tasks from Meta-World tasks.** The x-axis corresponds to the evaluation Δt (in ms), and the y-axis corresponds to success rate. We trained our TAWM model using either **RK4** or **Euler** integration method with log-uniform sampling strategy. The **baseline** method was trained using only the default time step ($\Delta t = 2.5ms$), which is signified with the black dashed lines. For fair comparisons, we extended the baseline method by **repeating** the same actions for the larger evaluation time steps than the default one. For instance, when the evaluation $\Delta t = 5ms$, we repeat the same action for $5/2.5 = 2$ times. Plots show mean and 95% confidence intervals over 3 seeds, with 10 evaluation episodes per seed.

B.5 Additional performance comparisons between TAWM and baseline models

In this section, we provide additional performance comparisons between TAWM and baseline models across varying inference Δt values, as shown in [fig. B.4](#). On each task, both TAWM variants (using RK4 and Euler integration methods) were trained for 1.5 million steps, with Δt values sampled from a Log-Uniform(1 ms, 50 ms) distribution during training. In contrast, the baseline models were trained solely on a fixed default time step of $\Delta t = 2.5$ ms. For these baselines, we use the pretrained weights provided by the original TD-MPC2 paper [71] for each corresponding task.

B.6 Additional learning curve comparisons between TAWM and baseline models

In this section, we present additional learning curve results for TAWM and baseline models on various Meta-World control tasks. Specifically, [fig. B.5](#) and [fig. B.6](#) show the learning curves of our time-aware models (TAWMs) and the non-time-aware baseline models (original TD-MPC2), evaluated at different inference Δt values. Each curve is generated by evaluating the models at various training steps using a fixed inference $\Delta t \in [1, 50]$ ms.

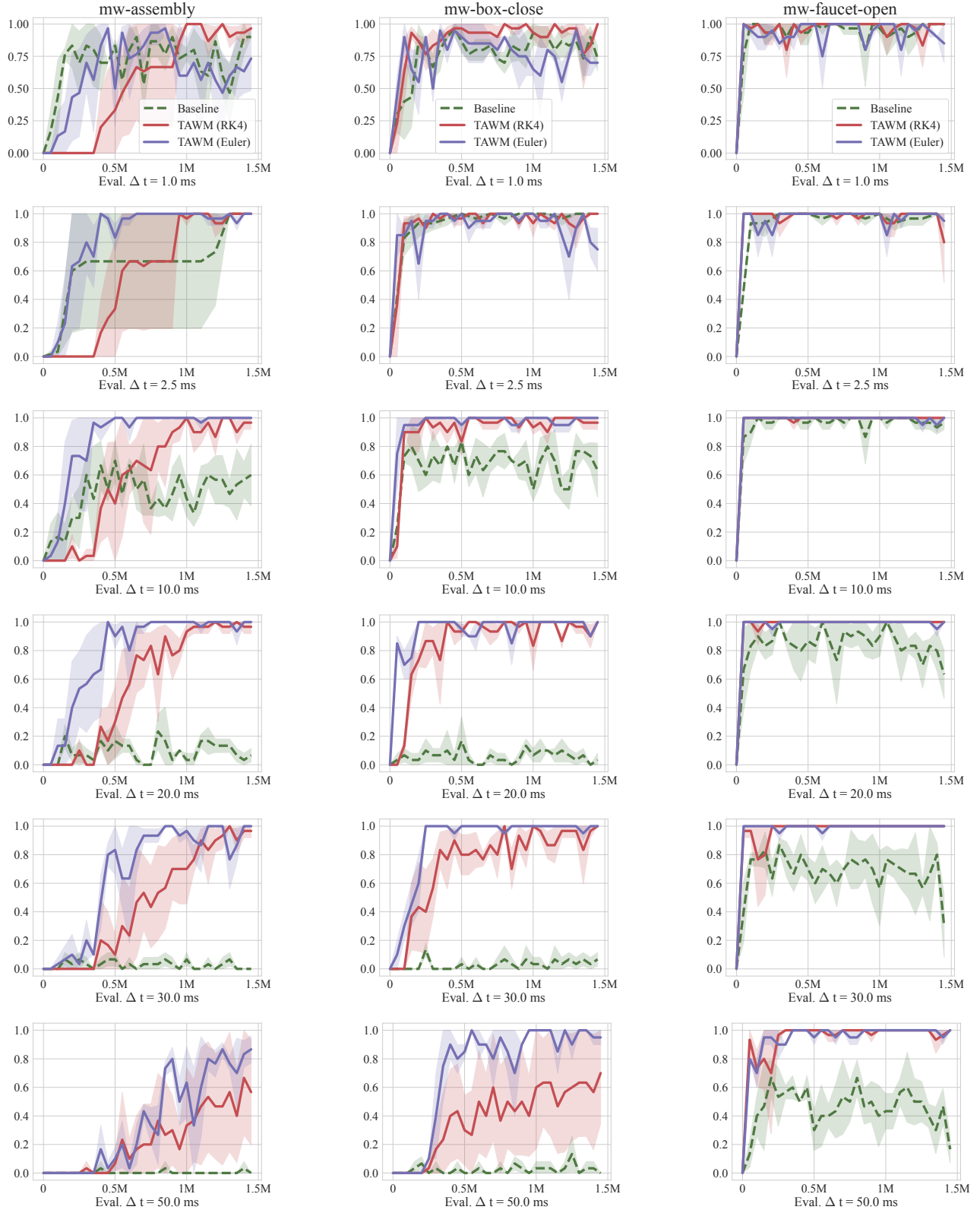


Figure B.5: Additional Meta-World tasks: Success Rate Curve under different evaluation time step sizes.

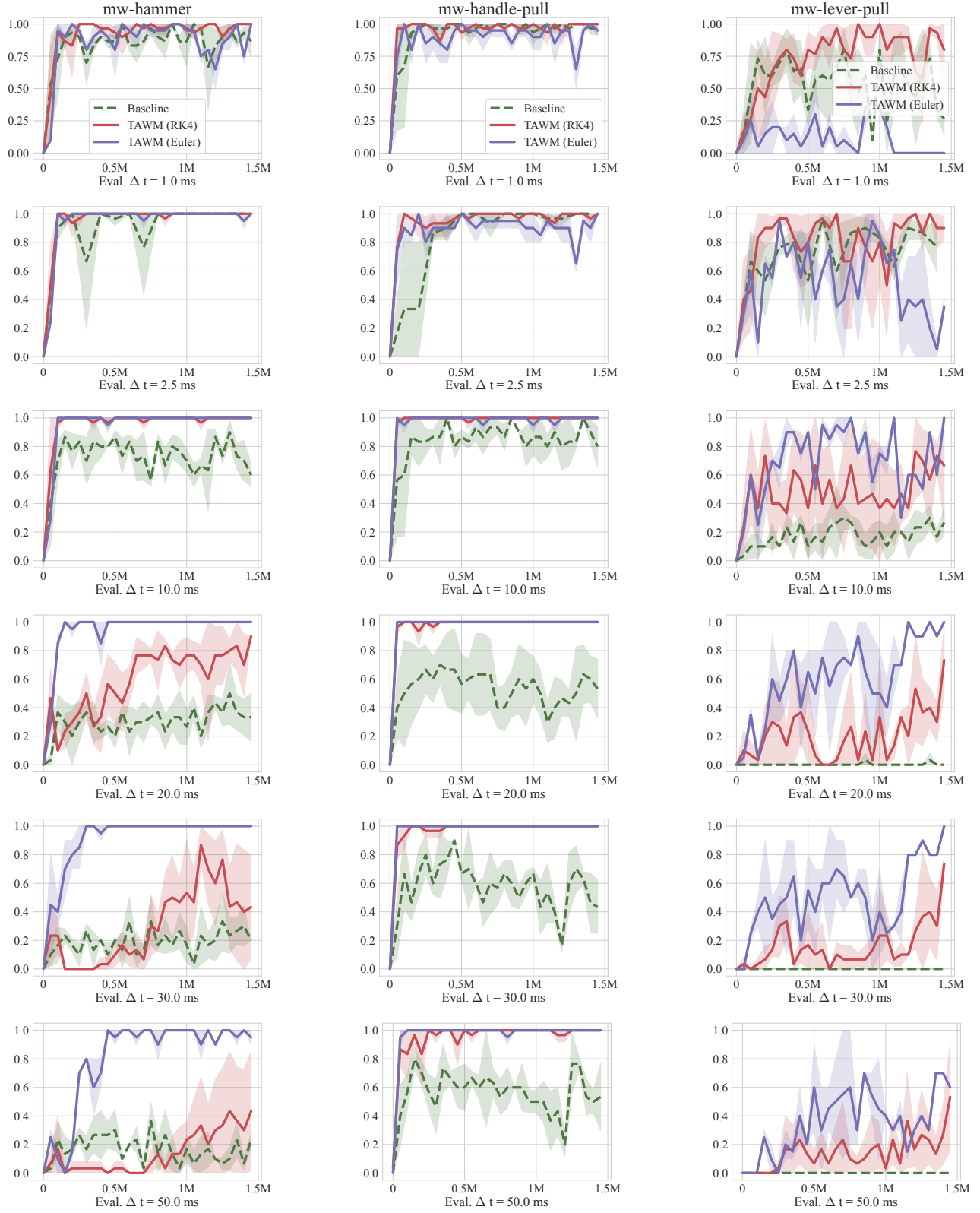


Figure B.6: Additional Meta-World tasks: Success Rate Curve under different evaluation time step sizes.

B.7 Additional experiments on PDE control problems.

In this section, we provide detailed descriptions of the PDE control tasks and present additional experiments and analysis of our TAWM on various PDE control problems in `control-gym` environments [198].

B.7.1 PDE Control Problems.

The PDE problems are one-dimensional PDE control problems featuring periodic boundary conditions and spatially distributed control inputs. The spatial domain is defined as $\Omega = [0, L] \subset \mathbb{R}$. The continuous field of the PDE is defined as $u(x, t) : \Omega \times \mathbb{R}^+ \rightarrow \mathbb{R}$, where x, t represent the spatial coordinates in the field and time, respectively. Generally, the PDE in each control task is defined as:

$$\frac{\partial u}{\partial t} - \mathcal{F} \left(\frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, \dots \right) = a(x, t)$$

In the equation above, $\mathcal{F}$ is a differential operator (linear or non-linear) defined differently for each control task. The specific formulations of $\mathcal{F}$ are carefully described in the next subsection for each environment. $a(x, t)$ is the distributed control force over the PDE field. $a(x, t)$ is defined as:

$$a(x, t) = \sum_{j=0}^{n_a-1} \Phi_j(x) a_j(t)$$

The control force is composed of n_a scalar control inputs $a_j(t)$, each influencing a specific subset of the domain Ω through its corresponding forcing support function $\Phi_j(x)$. Generally, each action introduces external forces/energy to the PDE fields to control their dynamics and steer them

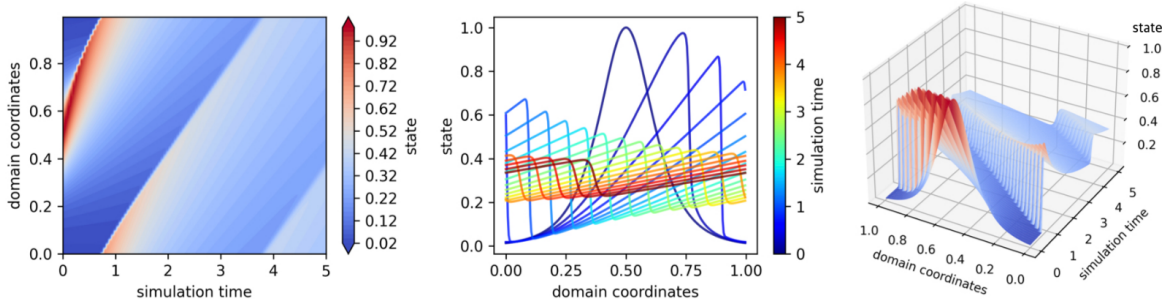


Figure B.7: Visualizations of systems dynamics of uncontrolled Burgers PDE. The spatial domain has length $L = 1$ with the diffusivity (viscosity) parameter $\nu = 10^{-3}$. The initial state is $u(x, t = 0) = \text{sech}(10x - 5)$.

toward the target state. More comprehensive details of the PDE environments are described in [198].

In our experiments, the target state of all PDE control tasks is $s_{\text{target}} = \vec{0}$. At each step t , the reward is computed as the LQ-error between the current state s_t and the target state s_{target} :

$$r_t = J(a_t) = -\mathbb{E} \left[(s_t - s_{\text{target}})^\top Q (s_t - s_{\text{target}}) + a_t^\top R a_t \right]$$

B.7.2 Burgers' Equation.

Burgers' equation is a simplified PDE that captures the essential dynamics of fluid waves and gas dynamics. The visualization of the Burgers' Equation is shown in Figure B.7. The field velocity $u(x, t)$ in Burgers' equation is defined as:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2} = a(x, t)$$

where $\nu > 0$ is the diffusivity (viscosity) parameter, and $a(x, t)$ is a source term modeling an external force acting on the PDE system at coordinate x and time t .

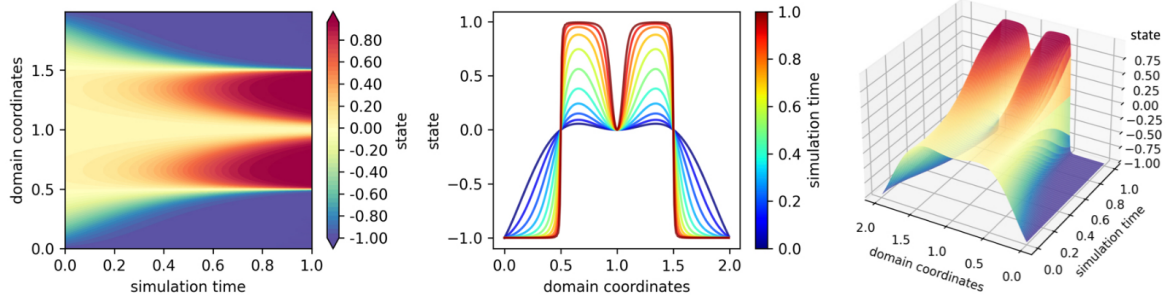


Figure B.8: Visualizations of systems dynamics of uncontrolled Allen-Cahn PDE. The spatial domain has length $L = 2$ with the diffusivity (viscosity) parameter $\nu = 10^{-4}$ and potential constant $V = 5.0$. The initial state is $u(x, t = 0) = (x - 1)^2 \cdot \cos(\pi(x - 1))$.

B.7.3 Allen-Cahn Equation.

In material sciences, the Allen-Cahn PDE is a non-linear PDE used to model the binary alloy systems' phase separation. The visualization of the Allen-Cahn equation is shown in Figure B.8. The temporal dynamics of field $u(x, t)$ is:

$$\frac{\partial u}{\partial t} - \nu^2 \frac{\partial^2 u}{\partial x^2} + V \cdot (u^3 - u) = a(x, t)$$

where $u = \pm 1$ indicates the presence of each phase, $\nu > 0$ is the diffusivity/viscosity parameter, V is the potential constant, and $a(x, t)$ is a source term modeling an external force acting on the PDE system at coordinate x and time t .

B.7.4 Wave Equation.

The wave equation is a second-order linear PDE describing the spatial propagation of waves in homogeneous mediums. Such wave propagation PDE has many important applications in physics and engineering problems. The visualization of Wave PDE is shown in Figure B.9. The scalar quantity

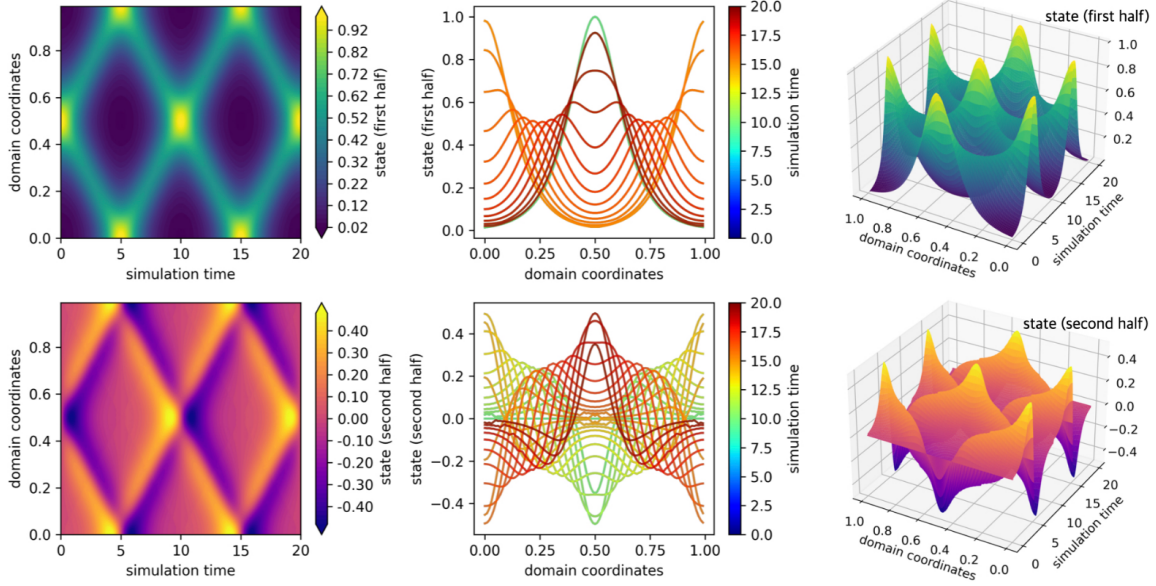


Figure B.9: Visualizations of systems dynamics of uncontrolled Wave PDE. The spatial domain has length $L = 1$ with $c = 0.1$. The initial state is $u(x, t = 0) = \text{sech}(10x - 5)$ and $\psi(x, t = 0) = 0$.

$u(x, t)$ has wave-propagation dynamics defined as:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = a(x, t)$$

where c is spatial wave speed, and $a(x, t)$ is a source term of a force acting on the system at coordinate x and time t .

B.7.5 Additional learning curve comparisons on PDE control tasks.

We present additional learning curves on PDE control tasks to provide insights into TAWM training efficiency on these tasks. Due to the small simulation time step required for stable simulation in PDE-Burgers ($\Delta t_{\text{sim}} = 10^{-3}$ seconds), training on this task takes longer in wall-clock time. Therefore, we trained TAWM for only 750k steps on PDE-Burgers. The learning curves of TAWM and the baseline under different observation Δt values are shown in Figure B.10. For each model and environment,

the same set of intermediate weights is used for evaluation at each training step.

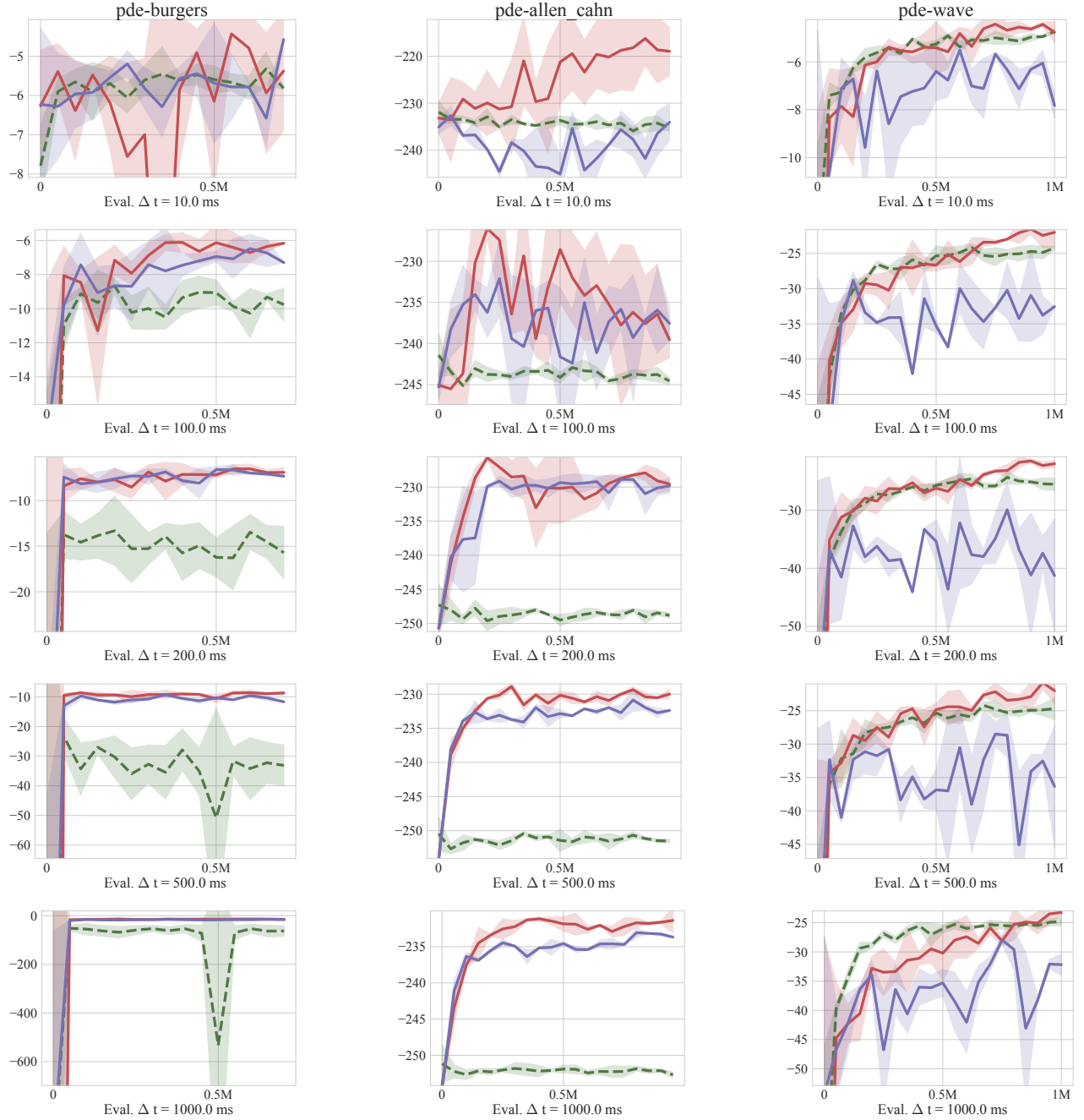


Figure B.10: PDE control tasks: Average Negative LQ Error learning curves under different evaluation time step sizes. The default time step size are $\Delta t = 50$ ms/10 ms/100 ms for PDE-Burgers, PDE-Allen-Cahn, PDE-Wave, respectively.

B.7.6 Proofs

In this section, we present the proofs of Lemma 3.4.1 and Lemma 3.4.2 from the theoretical analysis of TAWM's sample efficiency in Section 3.4.3.

B.7.6.1 Proof of Lemma 3.4.1

According to the definitions of the true dynamics function $\bar{f}$ and the optimal learned dynamics function d^* , we have

$$\begin{aligned} z_{t+\Delta t} - z_t &= \bar{f}(z_t, a_t, \Delta t) \Delta t = d^*(z_t, a_t, \Delta t) \tau(\Delta t) \\ \Rightarrow \bar{f}(z_t, a_t, \Delta t) &= d^*(z_t, a_t, \Delta t) \frac{\tau(\Delta t)}{\Delta t}. \end{aligned} \tag{B.2}$$

Since we are considering scenarios where the environment dynamics can be captured with $\Delta \bar{t}$, the following holds:

$$\begin{aligned} &\left\| d^*(z_t, a_t, \Delta t) \cdot \frac{\tau(\Delta t)}{\Delta t} - d^*(z_t, a_t, \Delta \bar{t}) \cdot \frac{\tau(\Delta \bar{t})}{\Delta \bar{t}} \right\| < \epsilon \\ \Rightarrow &\left\| d^*(z_t, a_t, \Delta t) - d^*(z_t, a_t, \Delta \bar{t}) \cdot \frac{\Delta t}{\tau(\Delta t)} \cdot \frac{\tau(\Delta \bar{t})}{\Delta \bar{t}} \right\| < \epsilon \cdot \frac{\Delta t}{\tau(\Delta t)}. \end{aligned} \tag{B.3}$$

Given the sampling-frequency assumption in Section 3.4.3 ($\Delta t \in [0.001, 1.0]$) and the definition of $\tau(\cdot)$, we observe that $\frac{\Delta t}{\tau(\Delta t)} < 1$. Consequently, Lemma 3.4.1 holds.

B.7.6.2 Proof of Lemma 3.4.2

Under Assumption 3 in Section 3.4.3, the following holds for our current dynamics model d :

$$\|d(z_t, a_t, \Delta t) - d(z_t, a_t, \Delta \bar{t}) \cdot \frac{\Delta t}{\tau(\Delta t)} \cdot \frac{\tau(\Delta \bar{t})}{\Delta \bar{t}}\| < \epsilon. \quad (\text{B.4})$$

If we denote $I(\Delta t) = \frac{\Delta t}{\tau(\Delta t)} \cdot \frac{\tau(\Delta \bar{t})}{\Delta \bar{t}}$, then the following holds:

$$\begin{aligned} & \|d(z_t, a_t, \Delta t) - d^*(z_t, a_t, \Delta t)\| \\ = & \|d(z_t, a_t, \Delta t) - d(z_t, a_t, \Delta \bar{t}) \cdot I(\Delta t) + d(z_t, a_t, \Delta \bar{t}) \cdot I(\Delta t) \\ & - d^*(z_t, a_t, \Delta t) + d^*(z_t, a_t, \Delta \bar{t}) \cdot I(\Delta t) - d^*(z_t, a_t, \Delta \bar{t}) \cdot I(\Delta t)\| \\ \leq & \|d(z_t, a_t, \Delta t) - d(z_t, a_t, \Delta \bar{t}) \cdot I(\Delta t)\| \\ & + \|d^*(z_t, a_t, \Delta t) - d^*(z_t, a_t, \Delta \bar{t}) \cdot I(\Delta t)\| \\ & + \|d(z_t, a_t, \Delta \bar{t}) - d^*(z_t, a_t, \Delta \bar{t})\| \cdot I(\Delta t) \\ = & 2\epsilon + \|d(z_t, a_t, \Delta \bar{t}) - d^*(z_t, a_t, \Delta \bar{t})\| \cdot I(\Delta t). \end{aligned} \quad (\text{B.5})$$

Likewise, the modeling error at time step Δt is upper bounded by the error at $\Delta \bar{t}$. Therefore, by minimizing the error at $\Delta \bar{t}$, we expect that the error at the smaller time step would safely decrease as well.

Appendix C: Differentiable Hybrid Traffic Simulation

C.1 Solution to the Riemann problem in ARZ model

C.1.1 Formulation

For the given left state $\mathbf{q}_l$ and right state $\mathbf{q}_r$, we can determine the intermediate state $\mathbf{q}_0$ as follows.

Case 0 : $u_l = u_r$

$$q_0 = q_l \tag{C.1}$$

Case 1 : $u_l > u_r$

$$q_0 = \begin{cases} q_l & \text{if } \lambda_s \geq 0 \\ q_m & \text{if } \lambda_s < 0 \end{cases} \quad (\lambda_s = \frac{\rho_m u_m - \rho_l u_l}{\rho_m - \rho_l}) \tag{C.2}$$

Case 2 : $u_r - u_{max} \rho_l^\gamma < u_l < u_r$

$$q_0 = \begin{cases} q_l & \text{if } \lambda_{0l} \geq 0 \\ q_m & \text{if } \lambda_{0m} \leq 0 \\ \tilde{q}(0) & \text{if } \lambda_{0l} < 0, \lambda_{0m} > 0 \end{cases} \tag{C.3}$$

where

$$\lambda_{0l} = u_l - u_{max} \gamma \rho_l^\gamma$$

$$\begin{aligned} \lambda_{0m} &= u_m - u_{max} \gamma \rho_m^\gamma \\ &= u_r - u_{max} \gamma \rho_l^\gamma + \gamma(u_r - u_l). \end{aligned}$$

Case 3 : $u_l \leq u_r - u_{max} \rho_l^\gamma$

$$q_0 = \begin{cases} q_l & \text{if } \lambda_{0l} \geq 0 \\ \tilde{q}(0) & \text{if } \lambda_{0l} < 0 \end{cases} \quad (\text{C.4})$$

Case 4 : $\rho_l = 0, \rho_r > 0$

$$q_0 = q_l = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (\text{C.5})$$

Case 5 : $\rho_l > 0, \rho_r = 0$

Same as Case 3.

From above, q_m is given as

$$\rho_m = \left(\rho_l^\gamma + \frac{u_l - u_r}{u_{max}} \right)^{\frac{1}{\gamma}}$$

$$u_m = u_r$$

$$y_m = \rho_m(u_m - u_{eq}(\rho_m)),$$

and $\tilde{q}(0)$ is given as

$$\begin{aligned}\tilde{\rho}(0) &= \left(\frac{u_l + u_{max}\rho_l^\gamma}{(\gamma + 1)u_{max}} \right)^{\frac{1}{\gamma}} \\ \tilde{u}(0) &= \frac{\rho_l}{\rho_l + 1} (u_l + u_{max}\rho_l^\gamma) \\ \tilde{y}(0) &= \tilde{\rho}(0)(\tilde{u}(0) - u_{eq}(\tilde{\rho}(0))).\end{aligned}$$

C.1.2 Differentiation

We can see from the above formulation that we have to compute the partial derivatives of q_l, q_m , and $\tilde{q}(0)$ with respect to q_l and q_r , as those values comprise the solutions of the Riemann problem. We can compute the partial derivatives as shown below.

The Jacobian of $\mathbf{f}$ is:

$$\mathbf{f}'(\mathbf{q}) = \begin{bmatrix} u_{eq} + \rho u'_{eq} & 1 \\ y u'_{eq} - \frac{y^2}{\rho^2} & \frac{2y}{\rho} + u_{eq} \end{bmatrix} \quad (\text{C.6})$$

Partial derivatives of $\mathbf{q}_l$

$$\frac{\partial q_l}{\partial q_l} = I_{2 \times 2}$$

$$\frac{\partial q_l}{\partial q_r} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Partial derivatives of q_m

$$\frac{\partial q_m}{\partial q_l} = \begin{bmatrix} \frac{\partial \rho_m}{\partial \rho_l} & \frac{\partial \rho_m}{\partial y_l} \\ \frac{\partial y_m}{\partial \rho_l} & \frac{\partial y_m}{\partial y_l} \end{bmatrix}$$

where

$$\begin{aligned} \frac{\partial \rho_m}{\partial \rho_l} &= \frac{1}{\gamma} \left(\rho_l^\gamma + \frac{u_l - u_r}{u_{max}} \right)^{\frac{1-\gamma}{\gamma}} \left(\gamma \rho_l^{\gamma-1} + \frac{1}{u_{max}} \frac{\partial u_l}{\partial \rho_l} \right), \\ \frac{\partial \rho_m}{\partial y_l} &= \frac{1}{\gamma} \left(\rho_l^\gamma + \frac{u_l - u_r}{u_{max}} \right)^{\frac{1-\gamma}{\gamma}} \left(\frac{1}{u_{max}} \frac{\partial u_l}{\partial y_l} \right), \\ \frac{\partial y_m}{\partial \rho_l} &= \frac{\partial \rho_m}{\rho_l} (u_m - u_{eq}(\rho_m)) + \rho_m (-u'_{eq}(\rho_m) \frac{\partial \rho_m}{\partial \rho_l}), \\ \frac{\partial y_m}{\partial y_l} &= \frac{\partial \rho_m}{y_l} (u_m - u_{eq}(\rho_m)) + \rho_m (-u'_{eq}(\rho_m) \frac{\partial \rho_m}{\partial y_l}). \end{aligned}$$

$$\frac{\partial q_m}{\partial q_r} = \begin{bmatrix} \frac{\partial \rho_m}{\partial \rho_r} & \frac{\partial \rho_m}{\partial y_r} \\ \frac{\partial y_m}{\partial \rho_r} & \frac{\partial y_m}{\partial y_r} \end{bmatrix}$$

where

$$\begin{aligned}
\frac{\partial \rho_m}{\partial \rho_r} &= \frac{1}{\gamma} \left(\rho_l^\gamma + \frac{u_l - u_r}{u_{max}} \right)^{\frac{1-\gamma}{\gamma}} \left(-\frac{1}{u_{max}} \frac{\partial u_r}{\partial \rho_r} \right), \\
\frac{\partial \rho_m}{\partial y_r} &= \frac{1}{\gamma} \left(\rho_l^\gamma + \frac{u_l - u_r}{u_{max}} \right)^{\frac{1-\gamma}{\gamma}} \left(-\frac{1}{u_{max}} \frac{\partial u_r}{\partial y_r} \right), \\
\frac{\partial y_m}{\partial \rho_r} &= \frac{\partial \rho_m}{\rho_r} (u_m - u_{eq}(\rho_m)) + \rho_m \left(\frac{\partial u_r}{\partial \rho_r} - u'_{eq}(\rho_m) \frac{\partial \rho_m}{\partial \rho_r} \right), \\
\frac{\partial y_m}{\partial y_r} &= \frac{\partial \rho_m}{y_r} (u_m - u_{eq}(\rho_m)) + \rho_m \left(\frac{\partial u_r}{\partial y_r} - u'_{eq}(\rho_m) \frac{\partial \rho_m}{\partial y_r} \right).
\end{aligned}$$

Partial derivatives of $\tilde{q}(0)$

$$\frac{\partial \tilde{q}(0)}{\partial q_l} = \begin{bmatrix} \frac{\partial \tilde{\rho}(0)}{\partial \rho_l} & \frac{\partial \tilde{\rho}(0)}{\partial y_l} \\ \frac{\partial \tilde{y}(0)}{\partial \rho_l} & \frac{\partial \tilde{y}(0)}{\partial y_l} \end{bmatrix}$$

where

$$\begin{aligned}
\frac{\partial \tilde{\rho}(0)}{\partial \rho_l} &= \frac{1}{\gamma} \left(\frac{u_l + u_{max} \rho_l^\gamma}{(\gamma + 1) u_{max}} \right)^{\frac{1-\gamma}{\gamma}} \left(\frac{1}{(\gamma + 1) u_{max}} \right) \left(\frac{\partial u_l}{\partial \rho_l} + u_{max} \gamma \rho_l^{\gamma-1} \right), \\
\frac{\partial \tilde{\rho}(0)}{\partial y_l} &= \frac{1}{\gamma} \left(\frac{u_l + u_{max} \rho_l^\gamma}{(\gamma + 1) u_{max}} \right)^{\frac{1-\gamma}{\gamma}} \left(\frac{1}{(\gamma + 1) u_{max}} \right) \left(\frac{\partial u_l}{\partial y_l} \right), \\
\frac{\partial \tilde{y}(0)}{\partial \rho_l} &= \frac{\partial \tilde{\rho}(0)}{\partial \rho_l} (\tilde{u}(0) - u_{eq}(\tilde{\rho}(0))) + \tilde{\rho}(0) \left(\frac{\partial \tilde{u}(0)}{\partial \rho_l} - u'_{eq}(\tilde{\rho}(0)) \frac{\partial \tilde{\rho}(0)}{\partial \rho_l} \right), \\
\frac{\partial \tilde{y}(0)}{\partial y_l} &= \frac{\partial \tilde{\rho}(0)}{\partial y_l} (\tilde{u}(0) - u_{eq}(\tilde{\rho}(0))) + \tilde{\rho}(0) \left(\frac{\partial \tilde{u}(0)}{\partial y_l} - u'_{eq}(\tilde{\rho}(0)) \frac{\partial \tilde{\rho}(0)}{\partial y_l} \right).
\end{aligned}$$

$$\frac{\partial \tilde{q}(0)}{\partial q_r} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

C.2 Differentiating IDM

C.2.1 Hyperparameters

In the IDM, following hyperparameters are used to describe different behavioral traits of discrete vehicles [109].

- s_{min} : Minimum desired distance to the leading vehicle.
- T_{pref} : Desired time to move forward with current speed.
- a_{max} : Upper bound of the computed acceleration.
- a_{pref} : Comfortable braking deceleration, positive value.
- v_{targ} : Target velocity it wants to maintain.
- $length_i$: Length of the i -th vehicle.

C.2.2 Differentiation

First we compute the gradients of $\mathbf{a}_i(t)$ with respect to $\mathbf{p}_i(t)$, $\mathbf{v}_i(t)$, $\mathbf{p}_{h(i)}(t)$, and $\mathbf{v}_{h(i)}(t)$, where $h(i)$ -th vehicle is the leading vehicle of i -th vehicle.

$$\begin{aligned}\frac{\partial \mathbf{a}_i(t)}{\partial \mathbf{p}_i(t)} &= -2a_{max} \frac{s_{opt}^2}{(p_{h(i)}(t) - p_i(t) - length_{h(i)})^3}, \\ \frac{\partial \mathbf{a}_i(t)}{\partial \mathbf{p}_{h(i)}(t)} &= 2a_{max} \frac{s_{opt}^2}{(p_{h(i)}(t) - p_i(t) - length_{h(i)})^3}, \\ \frac{\partial \mathbf{a}_i(t)}{\partial \mathbf{v}_i(t)} &= a_{max} \left(-\delta \frac{v_i(t)^{\delta-1}}{v_{targ}^\delta} - 2 \frac{s_{opt}}{(p_{h(i)}(t) - p_i(t) - length_{h(i)})^2} \frac{\partial s_{opt}}{\partial v_i(t)} \right), \\ \frac{\partial \mathbf{a}_i(t)}{\partial \mathbf{v}_{h(i)}(t)} &= a_{max} \left(-2 \frac{s_{opt}}{(p_{h(i)}(t) - p_i(t) - length_{h(i)})^2} \frac{\partial s_{opt}}{\partial v_{h(i)}(t)} \right),\end{aligned}$$

where

$$\begin{aligned}\frac{\partial s_{opt}}{\partial v_i(t)} &= T_{pref} + \frac{2v_i(t) - v_{h(i)}(t)}{2\sqrt{a_{max}a_{pref}}}, \\ \frac{\partial s_{opt}}{\partial v_{h(i)}(t)} &= \frac{-v_i(t)}{2\sqrt{a_{max}a_{pref}}}.\end{aligned}$$

Now for arbitrary i and j , we can compute the gradient of the state of the i -th vehicle with respect to that of the j -th vehicle by plugging in the above values. Note that the gradient is non-zero only when j equals to i or $h(i)$.

$$\frac{\partial \mathbf{p}_i(t+1)}{\partial \mathbf{p}_j(t)} = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{if } j = h(i) \\ 0 & \text{if } else \end{cases} \quad (\text{C.7})$$

$$\frac{\partial \mathbf{p}_i(t+1)}{\partial \mathbf{v}_j(t)} = \begin{cases} \Delta t & \text{if } j = i \\ 0 & \text{if } j = h(i) \\ 0 & \text{if } else \end{cases} \quad (\text{C.8})$$

$$\frac{\partial \mathbf{v}_i(t+1)}{\partial \mathbf{p}_j(t)} = \begin{cases} \Delta t(-2a_{max} \frac{s_{opt}^2}{(p_{h(i)}(t)-p_i(t)-length_{h(i)})^3}) & \text{if } j = i \\ \Delta t(2a_{max} \frac{s_{opt}^2}{(p_{h(i)}(t)-p_i(t)-length_{h(i)})^3}) & \text{if } j = h(i) \\ 0 & \text{if } else \end{cases} \quad (\text{C.9})$$

$$\frac{\partial \mathbf{v}_i(t+1)}{\partial \mathbf{v}_j(t)} = \begin{cases} 1 + \Delta t a_{max}(-\delta \frac{v_i(t)^{\delta-1}}{v_{targ}^{\delta}} - 2 \frac{s_{opt}}{(p_{h(i)}(t)-p_i(t)-length_{h(i)})^2} (T_{pref} + \frac{2v_i(t)-v_j(t)}{2\sqrt{a_{max}a_{pref}}})) & \text{if } j = i \\ \Delta t a_{max}(-2 \frac{s_{opt}}{(p_{h(i)}(t)-p_i(t)-length_{h(i)})^2} \frac{-v_i(t)}{2\sqrt{a_{max}a_{pref}}}) & \text{if } j = h(i) \\ 0 & \text{if } else \end{cases} \quad (\text{C.10})$$

Appendix D: Gradient-based Trajectory Optimization with Parallelized Differentiable Traffic Simulation

D.1 Feature Comparison with Existing Vehicle Simulators

Simulator	Parallelized	GPU Acceleration	ODE-Differentiable	Log Replay
HighwayEnv [116]	✗	✗	✗	✗
SUMO [312]	✗	✗	✗	✗
Waymax [124]	✓	✓	✗	✓
MOSS [122]	✓	✓	✗	✗
GPUDrive [125]	✓	✓	✗	✓
Ours	✓	✓	✓	✓

Table D.1: Feature comparisons between existing popular traffic simulators and ours. Note that this comparison is for vehicle simulation only.

Appendix E: DMesh: A Differentiable Mesh Representation

E.1 Comparison to Other Shape Reconstruction Methods

Here we provide conceptual comparisons between our approach and the other optimization-based 3D reconstruction algorithms, which use different shape representations. To be specific, we compared our method with mesh optimization methods starting from template mesh [3, 4], methods based on neural signed distance fields (SDF) [5, 6], methods based on neural unsigned distance fields (UDF) [7, 8], and methods based on differentiable isosurface extraction [9–12]. We used following criteria to compare these methods.

- Closed surface: Whether or not the given method can reconstruct, or represent closed surfaces.
- Open surface: Whether or not the given method can reconstruct, or represent open surfaces.
- Differentiable Meshing: Whether or not the given method can produce gradients from the loss computed on the final mesh.
- Differentiable Rendering: Whether or not the given method can produce gradients from the loss computed on the rendering results.
- Geometric topology: Whether or not the given method can change geometric topology of the shape. Here, geometric topology defines the continuous deformation of Euclidean subspaces [313].

For instance, genus of the shape is one of the traits that describe geometric topology.

Table E.1: Traits of different optimization-based shape reconstruction methods. We compare methods based on template mesh [3, 4], neural SDF [5, 6], neural UDF [7, 8], differentiable isosurface extraction techniques [9–12] with ours.

Methods	Closed	Open	Diff. Mesh	Diff. Render.	Geo. Topo.	Mesh Topo.	Manifold
Template Mesh	O	O	O	O	X	X	O
Neural SDF	O	X	X	O	O	X	O
Neural UDF	O	O	X	O	O	X	$\triangle$
Diff. Isosurface	O	$\triangle$	O	O	O	X	O
DMesh (Ours)	O	O	O	O	O	O	X

- Mesh topology: Whether or not the given method can produce gradients from the loss computed on the mesh topology, which denotes the structural configuration, or edge connectivity of a mesh.
- Manifoldness: Whether or not the given method guarantees manifold mesh.

In Table E.1, we present a comparative analysis of different methods. Note that our method meets all criteria, only except manifoldness. It is partially because our method does not assume volume, which is the same for methods based on neural UDF. However, because our method does not leverage smoothness prior of neural network like those methods, it could exhibit high frequency noises in the final mesh. Because of this reason, we gave $\triangle$ to the neural UDF methods, while giving X to our approach. When it comes to methods based on differentiable isosurface extraction algorithms, we gave $\triangle$ to its ability to handle open surfaces, because of [12]. They can represent open surfaces as subset of closed ones, but cannot handle non-orientable open surfaces. Finally, note that our method is currently the only method that can handle mesh topology.

Likewise, DMesh shows promise in addressing the shortcomings found in previous research. Nonetheless, it has its own set of limitations (Section 6.6). Identifying and addressing these limitations is crucial for unlocking the full potential of our method.

E.2 Details about Section 6.3.2

E.2.1 Mathematical Definitions

Here, we provide formal mathematical definitions of the terms used in Section 6.3.2. We mainly use notations from [266] and [282].

Generalizing the notations in Section 6.3.2, let $S[w]$ be a finite set of weighted points in $\mathbb{R}^d$, where w is a weight assignment that maps each point $p \in S$ to its weight w_p . We denote a weighted point as $p[w_p]$ and define power distance to measure distance between two weighted points.

Definition E.2.1 (Power distance) *Power distance between two weighted points $p[w_p]$ and $q[w_q]$ is measured as:*

$$\pi(p[w_p], q[w_q]) = d(p, q)^2 - w_p - w_q, \quad (\text{E.1})$$

where $d(p, q)$ is the Euclidean distance.

Note that an unweighted point is regarded as carrying weight of 0. Based on this power distance, we can define the power cell C_p of a point $p[w_p]$ as the set of unweighted points in $\mathbb{R}^d$ that are closer to $p[w_p]$ than to any other weighted point in $S[w]$.

Definition E.2.2 (Power cell) *Power cell of a point $p[w_p] \in S[w]$ is defined as:*

$$C_p = \{x \in \mathbb{R}^d \mid \forall q[w_q] \in S[w], \pi(x, p[w_p]) \leq \pi(x, q[w_q])\}. \quad (\text{E.2})$$

Note that some points may have empty power cells if their weights are relatively smaller than

neighboring points. We call them “submerged” points. As we will see later, weight regularization aims at submerging unnecessary points in mesh definition, which leads to mesh simplification.

To construct the power cell, we can use the concept of half space. A half space $H_{p<q}$ is the set of unweighted points in $\mathbb{R}^d$ that are closer to $p[w_p]$ than $q[w_q]$.

Definition E.2.3 (Half space) *Half space $H_{p<q}$ is defined as:*

$$H_{p<q} = \{x \in \mathbb{R}^d \mid \pi(x, p[w_p]) \leq \pi(x, q[w_q])\}. \quad (\text{E.3})$$

Note that we can construct a power cell by intersecting half spaces, which proves the convexity of the power cell. Now we call $H(p, q)$ as a half plane that divides $\mathbb{R}^d$ into two half spaces, $H_{p<q}$ and $H_{q<p}$.

Definition E.2.4 (Half plane) *Half plane $H_{p,q}$ is defined as:*

$$H_{p,q} = \{x \in \mathbb{R}^d \mid \pi(x, p[w_p]) = \pi(x, q[w_q])\}. \quad (\text{E.4})$$

Then, for a given k -simplex Δ^k comprised of weighted points $\{\Delta^k\} = \{p_1[w_{p_1}], \dots, p_{k+1}[w_{p_{k+1}}]\} \subset S[w]$, the dual structure D_{Δ^k} is the intersection of half planes between the points in $\{\Delta^k\}$, which is a convex set.

Definition E.2.5 (Dual form) *Dual form D_{Δ^k} of $\Delta^k = \{p_1[w_{p_1}], \dots, p_{k+1}[w_{p_{k+1}}]\}$ is defined as:*

$$D_{\Delta^k} = \bigcap_{i,j=1,\dots,k+1} H(p_i, p_j), \quad (\text{E.5})$$

which is equivalent to:

$$D_{\Delta^k} = \{x \in \mathbb{R}^d \mid \forall p_i[w_{p_i}], p_j[w_{p_j}] \in S[w], \pi(x, p_i[w_{p_i}]) = \pi(x, p_j[w_{p_j}])\}. \quad (\text{E.6})$$

Note that when $k = d$, D_{Δ^k} becomes a point, while it becomes a line when $k = d - 1$. As discussed in Section 6.3.2, we leverage the distance between this dual form D_{Δ^k} and reduced power cell [13] of the points in D_{Δ^k} to query the existence of Δ^k . The reduced power cell $R_{p|\Delta^k}$ is a power cell of p that does not concern the other points in Δ^k in its construction.

Definition E.2.6 (Reduced power cell) *Reduced power cell of a weighted point $p[w] \in S[w]$ for given Δ^k is defined as:*

$$R_{p|\Delta^k} = \{x \in \mathbb{R}^d \mid \forall q[w_q] \in S[w] - \{\Delta^k\}, \pi(x, p[w_p]) \leq \pi(x, q[w_q])\}. \quad (\text{E.7})$$

Using these concepts, we can measure the existence probability of a k -simplex, as provided in Section 6.3.2.

E.2.2 Analysis of previous approach

E.2.2.1 Theoretical Aspect

As mentioned in Section 6.3.2, the previous approach of [13] has two computational challenges in computational efficiency and precision. These limitations are mainly rooted in not knowing the power diagram structure before evaluating Eq. 6.3. We summarize the overall procedure of the previous approach, and point out how our method is different from it.

Collecting simplexes to evaluate First, we need to collect simplexes that we want to compute probabilities for. Here we assume the number of simplexes increases linearly ($O(N)$) as the number of points (N) increases. This is a plausible assumption, because we often search for k -nearest neighbors for each point, and combine them to generate the query simplexes.

Sampling a point on the dual forms of the simplexes The previous method relies on point projections to evaluate Eq. 6.3. This did not incur any problem for their case ($d = k = 2$), because the dual form was a single point. However, when ($k < d$) as in our cases, the dual form contains infinite number of points, which makes unclear how to apply this point-based approach. One possible solution is sampling the most “representative” point on the dual form, and leveraging the point to estimate Eq. 6.3. By definition, this estimation is lower bound of Eq. 6.3. However, the problem arises when this sample point does not give reliable result. For instance, we can consider the case shown in Figure 6.6(c). In the illustration, we can observe that Δ^1 exists in WDT, and thus D_{Δ^1} goes through $R_{p_1|D_{\Delta^1}}$. If we sample a point on D_{Δ^1} that is included in $R_{p_1|D_{\Delta^1}}$, the signed distance from the sample point would have same sign as Eq. 6.3. However, if the sample point is selected outside $R_{p_1|D_{\Delta^1}}$, the sign would be different from the real value. In this case, even if Δ^k exists, we can recognize it as not existing. Note that this false estimation produces false gradients, which could undermine optimization process.

In contrast, we do not have to concern about this precision issue, because we construct PD, which tells us good sample points that give reliable lower bounds of Eq. 6.3, when the given simplex exists in WDT. Otherwise, we explicitly compute minimum distance between the dual form and the reduced power cell, as discussed in Section 6.4.1.

Projecting sample points to reduced power cells The final step is point projection, where we project the sample points from dual forms to the reduced power cells to estimate Eq. 6.3. Based on the definitions in Appendix E.2, we can observe that a (reduced) power cell's boundaries are comprised of half planes. That is, the boundaries of C_p is comprised of multiple half planes between p and the other weighted points. However, when we do not know which half planes comprise the boundaries, we have to do exhaustive search to find the signed distance from the sample point to the boundaries of the reduced power cell. As the number of half planes that are associated with a point p is N , the computational cost to precisely compute the signed distance is $O(N)$.

Note that it does not hold for our case, because by constructing WDT and PD, we know which half planes form the boundaries of each power cell. Also, note that even when the number of points increases, the average number of half planes that comprise the boundaries of power cells remains constant. Therefore, in our case, this step requires only $O(1)$ computational cost.

Summary To sum up, the computational cost of the previous approach amounts to $O(N^2)$, as the number of simplexes to evaluate increases linearly, and the cost for the projection step also increases linearly as the number of points increase. However, the cost for ours remains at $O(N)$. Moreover, the previous approach does not guarantee satisfactory estimations of Eq. 6.3.

Before moving on, we point out that the original implementation limited the number of half planes to consider in evaluating Eq. 6.3 to reduce the computational cost to $O(N)$. This relaxation is permissible to the case where the precision is not very important. However, the precision is important in our case, because we aim at representing mesh accurately.

Table E.2: Comparison of computational speed and accuracy between the previous method [13] and ours. For each number of points in 2D, we report the triplet of (computational speed (ms) / false positive ratio (%) / false negative ratio (%)), along with the number of query simplexes that we give to the algorithm.

	# Pts.	100	300	1K	3K	10K	30K
(d=2, k=2)	Prev.	6.38 / 0 / 0	19.7 / 0 / 0	139 / 0 / 0	739 / 0 / 0.16	4376 / 0 / 0.12	-
	Ours.	105 / 0 / 0	107 / 0 / 0	108 / 0 / 0	111 / 0 / 0	116 / 0 / 0.12	125 / 0 / 0.25
	# Simp.	342	992	2274	3672	4974	6332
(d=2, k=1)	Prev.	5.95 / 0 / 49.9	16.9 / 0 / 49.8	116 / 0 / 49.9	609 / 0 / 49.9	3681 / 0 / 49.9	-
	Ours.	99.4 / 0 / 0	100 / 0 / 0	103 / 0 / 0	105 / 0.03 / 0	111 / 0 / 0.08	123 / 0 / 0.29
	# Simp.	526	1504	3428	5522	7482	9524

E.2.2.2 Experimental Results

To prove the aforementioned theoretical claim, we conducted experiments to measure the computational speed and accuracy of the probability estimation for $(d = 2, k = 2)$ and $(d = 2, k = 1)$ cases, for varying number of points. We randomly sampled points in a unit cube uniformly, and set the weights by random sampling from a normal distribution: $\mathcal{N}(0, 10^{-3})$. For fair comparison, we did not use CUDA implementation that we mainly use in the paper. We implemented both algorithms in PyTorch, and ran 10 times for each setting to get fair values. In each experiment, we fed the query simplexes into the algorithm, and computed their existence probabilities. If the computed probability for a simplex was over 0.5, but did not exist, it is counted to false positive ratio. In contrast, if the computed probability for a simplex was below 0.5, but did exist, it is counted to false negative ratio. We measured the computational accuracy with these metrics.

In Table E.2, we can see that as the number of points increases, the computational cost increases exponentially for the previous approach, while ours remain fairly stable up to 30K points. When the number of points reached 100K, the previous method failed because of excessive memory consumption. However, when the number of points is smaller than 1K, the previous method ran faster than

ours, because we need to construct PD even for the small number of points, which consumes most of the time for all these cases.

In terms of accuracy, we can observe that the previous method and ours both give accurate estimations when $(d = 2, k = 1)$. However, when $(d = 2, k = 1)$, we can see that the false negative ratio is almost 50% – as we set the number of existing simplexes and non-existing simplexes in the query set as the same, it means that the previous method chose wrong sample points for most of the query simplexes, so that it predicted most of them as not existing. To select the most representative point, we found the intersection point between the dual form and the affine space created by the weighted points in the simplex. This could be another interesting direction to explore, but this approach found out to be not very accurate in this experiment. In contrast, our approach gave stable estimations for all the cases. This result demonstrates that our method is more scalable, and gives reliable probability estimations than the previous approach, which is necessary for accurate 3D mesh representation.

E.3 Loss Functions

Here we provide formal definitions for the loss functions that we use in the paper.

E.3.1 Mesh to DMesh

In this section, we explore the loss function used to transform the ground truth mesh into our DMesh representation. As previously mentioned in Section 8.3.4, the explicit definition of ground truth connectivity in the provided mesh allows us to establish a loss function based on it.

Building on the explanation in Section 8.3.4, if the ground truth mesh consists of vertices $\mathbb{P}$ and faces $\mathbb{F}$, we can construct an additional set of faces $\bar{\mathbb{F}}$. These faces are formed from vertices in $\mathbb{P}$ but

do not intersect with faces in $\mathbb{F}$.

$$\bar{\mathbb{F}} = \mathbb{F}^* - \mathbb{F}, \text{ where } \mathbb{F}^* = \text{every possible face combination on } \mathbb{P}.$$

Then, we notice that we should maximize the existence probabilities of faces in $\mathbb{F}$, but minimize those of faces in $\bar{\mathbb{F}}$. Therefore, we can define our reconstruction loss function as

$$L_{recon} = - \sum_{F \in \mathbb{F}} \Lambda(F) + \sum_{F \in \bar{\mathbb{F}}} \Lambda(F). \quad (\text{E.8})$$

If the first term of the loss function mentioned above is not fully optimized, it could lead to the omission of ground truth faces, resulting in a poorer recovery ratio (Section 6.5.1). Conversely, if the second term is not fully optimized, the resulting DMesh might include faces absent in the ground truth mesh, leading to a higher false positive ratio (Section 6.5.1). Refer to Appendix E.4.1 for details on how this reconstruction loss is integrated into the overall optimization process.

E.3.2 Point Cloud Reconstruction

In the task of point cloud reconstruction, we reconstruct the mesh by minimizing the (L_1 -norm based) expected Chamfer Distance (CD) between the given point cloud ($\mathbb{P}_{gt}$) and the sample points ($\mathbb{P}_{ours}$) from our reconstructed mesh. We denote the CD from $\mathbb{P}_{gt}$ to $\mathbb{P}_{ours}$ as CD_{gt} , and the CD from $\mathbb{P}_{ours}$ to $\mathbb{P}_{gt}$ as CD_{ours} . The final reconstruction loss is obtained by combining these two distances.

$$L_{recon} = CD_{gt} + CD_{ours}. \quad (\text{E.9})$$

E.3.2.1 Sampling $\mathbb{P}_{ours}$

To compute these terms, we start by sampling $\mathbb{P}_{ours}$ from our current mesh. First, we sample a set of faces that we will sample points from. We consider the areas of the triangular faces and their existence probabilities. To be specific, we define $\eta(F)$ for a face F as

$$\bar{\eta}(F) = \Lambda(F), \quad \eta(F) = F_{area} \cdot \bar{\eta}(F),$$

and define a probability to sample F from the entire faces $\mathbb{F}$ as

$$P_{sample}(F) = \frac{\eta(F)}{\sum_{F' \in \mathbb{F}} \eta(F')}.$$

We sample N faces from $\mathbb{F}$ with replacement and then uniformly sample a single point from each selected face to define $\mathbb{P}_{ours}$. In our experiments, we set N to $100K$.

In this formulation, we sample more points from faces with a larger area and higher existence probability to improve sampling efficiency. However, we observed that despite these measures, the sampling efficiency remains low, leading to slow convergence. This issue arises because, during optimization, there is an excessive number of faces with very low existence probability.

To overcome this limitation, we decided to do stratified sampling based on point-wise real values and cull out faces with very low existence probabilities. To be specific, we define two different η

functions:

$$\begin{aligned}\bar{\eta}_1(F) &= \Lambda_{wdt}(F) \cdot \min(\psi_i, \psi_j, \psi_k), & \eta_1(F) &= F_{area} \cdot \bar{\eta}_1(F) \\ \bar{\eta}_2(F) &= \Lambda_{wdt}(F) \cdot \max(\psi_i, \psi_j, \psi_k), & \eta_2(F) &= F_{area} \cdot \bar{\eta}_2(F)\end{aligned}$$

where (ψ_i, ψ_j, ψ_k) are the real values of the points that comprise F . Note that η_1 is the same as η ¹.

For the faces in $\mathbb{F}$, we first calculate the $\bar{\eta}_1$ and $\bar{\eta}_2$ values and eliminate faces with values lower than a predefined threshold ϵ_η . We denote the set of remaining faces as $\mathbb{F}_1$ and $\mathbb{F}_2$. Subsequently, we sample $\frac{N}{2}$ faces from $\mathbb{F}_1$ and the other $\frac{N}{2}$ faces from $\mathbb{F}_2$, using the following two sampling probabilities:

$$P_{sample,1}(F) = \frac{\eta_1(F)}{\sum_{F' \in \mathbb{F}_1} \eta_1(F')}, \quad P_{sample,2}(F) = \frac{\eta_2(F)}{\sum_{F' \in \mathbb{F}_2} \eta_2(F')}.$$

The rationale behind this sampling strategy is to prioritize (non-existing) faces closer to the current mesh over those further away. In the original $\eta = \eta_1$ function, we focus solely on the minimum real value, leading to a higher sampling rate for existing faces. However, to remove holes in the current mesh, it's beneficial to sample more points from potential faces—those not yet existing but connected to existing ones. This approach, using η_2 , enhances reconstruction results by removing holes more effectively. Yet, there's substantial potential to refine this importance sampling technique, as we haven't conducted a theoretical analysis in this study.

Moreover, when sampling a point from a face, we record the face's existence probability alongside the point. Additionally, if necessary, we obtain and store the face's normal. For a point $\mathbf{p} \in \mathbb{P}_{ours}$,

¹We do not use differentiable min operator, as we do not require differentiability in the sampling process.

we introduce functions $\Lambda_{pt}(\cdot)$ and $Normal(\cdot)$ to retrieve the face existence probability and normal, respectively:

$$\Lambda_{pt}(\mathbf{p}) = \Lambda(F(\mathbf{p})), \quad Normal(\mathbf{p}) = F(\mathbf{p})_{normal},$$

$F(\mathbf{p})$ = the face where $\mathbf{p}$ was sampled from.

E.3.2.2 CD_{gt}

Now we introduce how we compute the CD_{gt} , which is CD from $\mathbb{P}_{gt}$ to $\mathbb{P}_{ours}$. For each point $\mathbf{p} \in \mathbb{P}_{gt}$, we first find k -nearest neighbors of $\mathbf{p}$ in $\mathbb{P}_{ours}$, which we denote as $(p_1, p_2, \dots, p_k)$. Then, we define a distance function between the point $\mathbf{p}$ and the k -nearest neighbors as follows, to accommodate the orientation information:

$$\bar{D}(\mathbf{p}, p_i) = \|\mathbf{p} - p_i\|_2 + \lambda_{normal} \cdot \bar{D}_n(\mathbf{p}, p_i), \quad (E.10)$$

where $\bar{D}_n(\mathbf{p}, p_i) = 1 - |< \mathbf{p}_{normal}, Normal(p_i) >|$,

where λ_{normal} is a parameter than determines the importance of point orientation in reconstruction. If $\lambda_{normal} = 0$, we only consider the positional information of the sampled points.

After we evaluate the above distance function values for the k -nearest points, we reorder them

in ascending order. Then, we compute the following expected minimum distance from $\mathbf{p}$ to $\mathbb{P}_{ours}$,

$$D(\mathbf{p}, \mathbb{P}_{ours}) = \sum_{i=1, \dots, k} \bar{D}(\mathbf{p}, p_i) \cdot P(p_i) \cdot \bar{P}(p_i),$$

$$P(p_i) = \Lambda_{pt}(p_i) \cdot \mathbb{I}_{prev}(F(p_i)),$$

$$\bar{P}(p_i) = \prod_{i=1, \dots, k-1} (1 - P(p_i)),$$

where $\mathbb{I}_{prev}$ is an indicator function that returns 1 only when the given face has not appeared before in computing the above expected distance. For instance, if the face ids for the reordered points were $(1, 2, 3, 2, 3, 4)$, the $\mathbb{I}_{prev}$ function evaluates to $(1, 1, 1, 0, 0, 1)$. This indicator function is needed, because if we select p_i as the nearest point to $\mathbf{p}$ with the probability $\Lambda_{pt}(\mathbf{p})$, it means that we interpret that the face corresponding to p_i already exists, and then we would select p_i on the face as the nearest point to $\mathbf{p}$ rather than the other points that were sampled from the same face, but have larger distance than p_i and thus come after p_i in the ordered points.

Note that we dynamically change k during runtime to get a reliable estimation of $D(\mathbf{p}, \mathbb{P}_{ours})$. That is, for current k , if most of $\bar{P}(p_k)$ s for the points in $\mathbb{P}_{gt}$ are still large, it means that there is a chance that the estimation could change a lot if we find and consider more neighboring points. Therefore, in our experiments, if any point in $\mathbb{P}_{gt}$ has $\bar{P}(p_k)$ larger than 10^{-4} , we increase k by 1 for the next iteration. However, if there is no such point, we decrease k by 1 to accelerate the optimization process.

Finally, we can compute CD_{gt} by summing up the point-wise expected minimum distances.

$$CD_{gt} = \sum_{\mathbf{p} \in \mathbb{P}_{gt}} D(\mathbf{p}, \mathbb{P}_{ours}).$$

E.3.2.3 CD_{ours}

In computing CD_{ours} , which is CD from $\mathbb{P}_{ours}$ to $\mathbb{P}_{gt}$, we also find k -nearest neighbors for each point $\mathbf{p} \in \mathbb{P}_{ours}$, which we denote as $(p_1, p_2, \dots, p_k)$. Then, for a point $\mathbf{p}$, we use the same distance function $\bar{D}$ in Eq. E.10 to find the distance between $\mathbf{p}$ and $(p_1, p_2, \dots, p_k)$. After that, we select the minimum one for each point, multiply the existence probability of each point, and then sum them up to compute CD_{ours} .

$$D(\mathbf{p}, \mathbb{P}_{gt}) = \min_{i=1, \dots, k} \bar{D}(\mathbf{p}, p_i),$$

$$CD_{ours} = \sum_{\mathbf{p} \in \mathbb{P}_{ours}} \Lambda_{pt}(\mathbf{p}) \cdot D(\mathbf{p}, \mathbb{P}_{gt}).$$

Finally, we can compute the final reconstruction loss for point clouds as shown in Eq. E.9.

E.3.3 Multi-View Reconstruction

When we are given multi-view images, we reconstruct the mesh by minimizing the L_1 difference between our rendered images and the given images. In this work, we mainly use both diffuse and depth renderings to reconstruct the mesh.

If we denote the (N_{img}) ground truth images of N_{pixel} number of pixels as $\mathcal{I}_i^{gt} (i = 1, \dots, N_{img})$, and our rendered images as $\mathcal{I}_i^{ours}$, we can write the reconstruction loss function as

$$L_{recon} = \frac{1}{N_{img} \cdot N_{pixel}} \sum_{i=1, \dots, N_{img}} \|\mathcal{I}_i^{gt} - \mathcal{I}_i^{ours}\|.$$

Then, we can define our rendered image as follows:

$$I_i^{ours} = \mathcal{F}(\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), \mathbf{MV}_i, \mathbf{P}_i).$$

where $\mathcal{F}$ is a differentiable renderer that renders the scene for the given points $\mathbb{P}$, faces $\mathbb{F}$, face existence probabilities $\Lambda(\mathbb{F})$, i -th modelview matrix $\mathbf{MV}_i \in \mathbb{R}^{4 \times 4}$, and i -th projection matrix $\mathbf{P}_i \in \mathbb{R}^{4 \times 4}$. The differentiable renderer $\mathcal{F}$ has to backpropagate gradients along $\mathbb{P}$, $\mathbb{F}$, and $\Lambda(\mathbb{F})$ to update our point attributes. Specifically, here we interpret $\Lambda(\mathbb{F})$ as opacity for faces to use in the rendering process. This is because opacity means the probability that a ray stops when it hits the face, which aligns with our face existence probability well. For this reason, we ignore faces with very low existence probability under some threshold to accelerate the reconstruction, as they are almost transparent and do not contribute to the rendering a lot.

To implement $\mathcal{F}$, we looked through previous works dedicated for differentiable rendering [24, 161]. However, we discovered that these methods incur substantial computational costs when rendering a large number of (potentially) semi-transparent triangles, as is the case in our scenario. Consequently, we developed two efficient, partially differentiable renderers that meet our specific requirements. These renderers fulfill distinct roles within our pipeline—as detailed in Appendix E.4, our optimization process encompasses two phases within a single epoch. The first renderer is employed during the initial phase, while the second renderer is utilized in the subsequent phase.

E.3.3.1 $\mathcal{F}_A$

If there are multiple semi-transparent faces in the scene, we have to sort the faces that covers a target pixel with their (view-space) depth values, and iterate through them until the accumulated trans-

mittance is saturated to determine the color for the pixel. Conducting this process for each individual pixel is not only costly, but also requires a lot of memory to store information for backward pass.

Recently, 3D Gaussian Splatting [280] overcame this issue with tile-based rasterizer. We adopted this approach, and modified their implementation to render triangular faces, instead of gaussian splats. To briefly introduce its pipeline, it first assigns face-wise depth value by computing the view-space depth of its center point. Then, after subdividing the entire screen into 16×16 tiles, we assign faces to each tiles if they overlap. After that, by using the combination of tile ID and the face-wise depth as a key, we get the face list sorted by depth value in each tile. Finally, for each tile, we iterate through the sorted faces and determine color and depth for each pixel as follows.

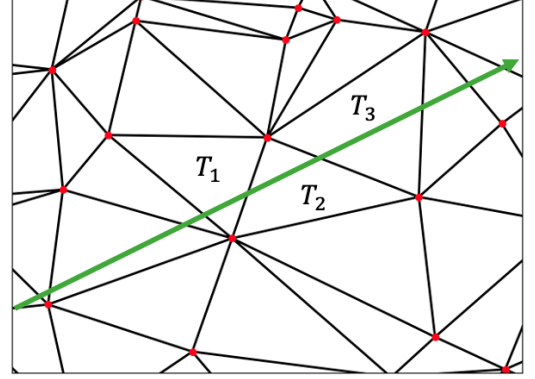


Figure E.1: $\mathcal{F}_B$ uses tessellation structure to efficiently render overlapped faces in the correct order.

$$C = \sum_{i=1, \dots, k} T_i \cdot \alpha_i \cdot C_i, \quad (T_i = \prod_{j=1, \dots, i-1} (1 - \alpha_j)),$$

where T_i is the accumulated transmittance, α_i is the opacity of the i -th face, and C_i is the color (or depth) of the i -th face. Note that $\alpha_i = \Lambda(F_i)$, as mentioned above.

Even though this renderer admits an efficient rendering of large number of semi-transparent faces, there are still two large limitations in the current implementation. First, the current implementation does not produce visibility-related gradients (near face edges) to update point attributes. Therefore, we argue that this renderer is partially differentiable, rather than fully differentiable. Next, since it does not compute precise view-point depth for each pixel, its rendering result can be misleading for

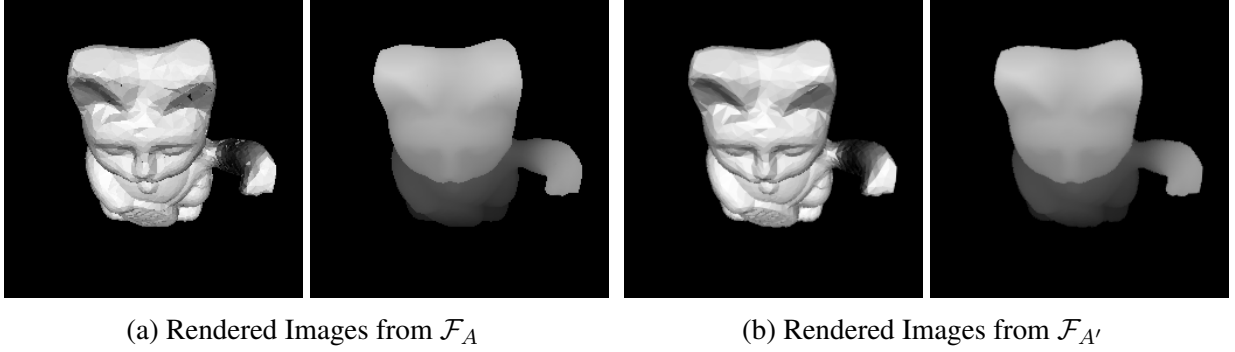


Figure E.2: Rendered images from two differentiable renderers, $\mathcal{F}_A$ and $\mathcal{F}_{A'}$. Left and right image corresponds to diffuse and depth rendering, respectively. (a) $\mathcal{F}_A$ is our (partially) differentiable renderer based on tile-based approach. (b) Since $\mathcal{F}_A$ does not produce visibility-related gradients, we additionally use $\mathcal{F}_{A'}$ [24] to render images and integrate with ours.

some cases, as pointed out in [280].

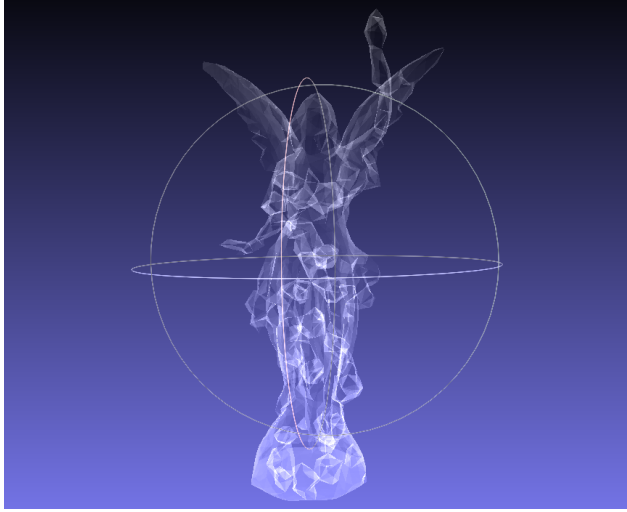
To amend the first issue, we opt to use another differentiable renderer of [24], which produces the visibility-related gradients that we lack. Since this renderer cannot render (large number of) transparent faces as ours does, we only render the faces with opacity larger than 0.5. Also, we set the faces to be fully opaque. If we call this renderer as $\mathcal{F}_{A'}$, our final rendered image can be written as follows.

$$\mathcal{I}_i^{ours} = \frac{1}{2}(\mathcal{F}_A(\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), \mathbf{M}\mathbf{V}_i, \mathbf{P}_i) + \mathcal{F}_{A'}(\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), \mathbf{M}\mathbf{V}_i, \mathbf{P}_i)).$$

In Figure E.2, we illustrate rendered images from $\mathcal{F}_A$ and $\mathcal{F}_{A'}$.

Acknowledging that this formulation is not theoretically correct, we believe that it is an intriguing future work to implement a fully differentiable renderer that works for our case. However, we empirically found out that we can reconstruct a wide variety of meshes with current formulation without much difficulty.

As mentioned before, this renderer is used at the first phase of the optimization process, where all of the point attributes are updated. However, in the second phase, we fix the point positions and



(a) Extracted Mesh after phase 1



(b) Extracted Mesh after phase 2

Figure E.3: Reconstructed mesh from multi-view images, rendered in MeshLab’s [25] x-ray mode to see inner structure. In multi-view reconstruction, we divide each epoch in two phases. (a) After the first phase ends, where we do inaccurate depth testing, lots of false inner faces are created. (b) To remove these inner faces, we require a renderer that does the exact depth testing, which we use in the second phase. Also see Appendix E.4.2 for details about post-processing step to remove the inner structure.

weights, and only update point-wise real values (Appendix E.4.2). In this case, we can leverage the tessellation structure to implement an efficient differentiable renderer. As the second renderer does a precise depth testing unlike the first one, it can be used to modify the errors incurred by the second limitation of the first renderer (Figure E.3).

E.3.3.2 $\mathcal{F}_B$

The second renderer performs precise depth ordering in an efficient way, based on the fixed tessellation structure that we have. In Figure E.1, we illustrate a 2D diagram that explains our approach. When the green ray, which corresponds to a single ray to determine the color of a single pixel, goes through the tessellation, we can observe that it goes through a sequence of triangles (tetrahedron in 3D), which are denoted as T_1, T_2 , and T_3 . When the ray enters a triangle T_i through one of its three

edges, we can see that it moves onto the other adjacent triangle T_{i+1} only through one of the other edges of T_i , because of compact tessellation. Therefore, when the ray hits one edge of T_i , it can only examine the other two edges of T_i to find the next edge it hits. Note that we do not have to do depth testing explicitly in this approach. Also, unlike the first approach, this renderer does not have to store all the possible faces that a ray collides for the backward pass, because it can iterate the same process in the opposite way in the backward pass to find the edge that it hit before the last edge. If we only store the last edge that each hits at the forward pass, we can start from the last edge and find the previous edges that it hit to compute gradients. Therefore, this second renderer requires much less memory than the first one, and also performs precise depth testing naturally. However, note that this renderer is also partially differentiable, because it cannot update point positions and weights.

To sum up, we implemented two partially differentiable renderers to solve multi-view reconstruction problem with DMesh. They serve different objectives in our reconstruction process, and we empirically found out that they are powerful enough to reconstruct target meshes in our experiments. However, we expect that we can simplify the process and improve its stability, if we can implement a fully differentiable renderer that satisfy our needs. We leave it as a future work.

E.3.4 Weight Regularization

Weight regularization aims at reducing the complexity of WDT, which supports our mesh. By using this regularization, we can discard unnecessary points that do not contribute to representing our mesh. Moreover, we can reduce the number of points on the mesh, if they are redundant, which ends up in the mesh simplification effect (Appendix [E.5.3](#)).

We formulate the complexity of WDT as the sum of edge lengths in its dual power diagram.

Formally, we can write the regularization as follows,

$$L_{weight} = \sum_{i=1,\dots,N} Length(E_i),$$

where E_i are the edges in the dual power diagram, and N is the number of edges.

E.3.5 Real Regularization

Real regularization is a regularization that is used for maintaining the real values of the connected points in WDT as similar as possible. Also, we leverage this regularization to make real values of points that are connected to the points with high real values to become higher, so that they can be considered in reconstruction more often than the points that are not connected to those points. To be specific, note that we ignore faces with very low existence probability in the reconstruction process. By using this regularization, it can remove holes more effectively.

This real regularization can be described as

$$\begin{aligned} L_{real} &= \frac{1}{\sum_{i=1,\dots,N} \Lambda(F_i)} \sum_{i=1,\dots,N} \Lambda(F_i) \cdot (\sigma_1(F_i) + \sigma_2(F_i)), \\ \sigma_1(F_i) &= \frac{1}{3} \sum_{j=1,2,3} \left| \psi_j - \frac{(\psi_1 + \psi_2 + \psi_3)}{3} \right|, \\ \sigma_2(F_i) &= \frac{1}{3} \sum_{j=1,2,3} |1 - \psi_j| \cdot \mathbb{I}(\max_{j=1,2,3} (\psi_j) > \delta_{high}). \end{aligned}$$

Here $\psi_{1,2,3}$ represent the real values of points that comprise F_i , and δ_{high} is a threshold to determine “high” real value, which is set as 0.8 in our experiments. Note that the faces with higher existence probabilities are prioritized over the others.

E.3.6 Quality Regularization

After reconstruction, we usually want to have a mesh that is comprised of triangles of good quality, rather than ill-formed triangles. We adopt the aspect ratio as a quality measure for the triangular faces, and minimize the sum of aspect ratios for all faces during optimization to get a mesh of good quality. Therefore, we can write the regularization as follows.

$$L_{qual} = \frac{1}{\sum_{i=1,\dots,N} \Lambda(F_i)} \sum_{i=1,\dots,N} AR(F_i) \cdot E_{max}(F_i) \cdot \Lambda(F_i),$$
$$AR(F_i) = \frac{E_{max}(F_i)}{H_{min}(F_i)} \cdot \frac{\sqrt{3}}{2},$$
$$E_{max}(F_i) = \text{Maximum edge length of } F_i,$$
$$H_{min}(F_i) = \text{Minimum height of } F_i.$$

Note that we prioritize faces with larger maximum edge length and higher existence probability than the others in this formulation. In Appendix [E.5.3](#), we provide ablation studies for this regularization.

E.4 Optimization Process

In this section, we explain the optimization processes, or exact reconstruction algorithms, in detail. First, we discuss the optimization process for the experiment in Section [6.5.1](#), where we represent the ground truth mesh with DMesh. Then, we discuss the overall optimization process for point cloud or multi-view reconstruction tasks in Section [6.5.2](#), from initialization to post processing.

E.4.1 Mesh to DMesh

Our overall algorithm to convert the ground truth mesh into DMesh is outlined in Algorithm 3.

We explain each step in detail below.

Algorithm 3 Mesh to DMesh

```

1:  $\mathbb{P}_{gt}, \mathbb{F}_{gt} \leftarrow$  Ground-truth mesh vertices and faces
2:  $\mathbb{P}, \mathbb{W}, \psi \leftarrow$  Initialize point attributes for DMesh
3:  $\bar{\mathbb{F}} \leftarrow \emptyset$ 
4: while optimization not ended do
5:   Perform point insertion using  $\mathbb{P}$  and  $\bar{\mathbb{F}}$ 
6:    $WDT, PD \leftarrow$  Run WDT with  $\mathbb{P}, \mathbb{W}$ 
7:    $\bar{\mathbb{F}} \leftarrow$  Update faces to exclude based on  $WDT$ 
8:    $\Lambda(\mathbb{F}_{gt}), \Lambda(\bar{\mathbb{F}}) \leftarrow$  Compute face existence probabilities using  $\mathbb{P}, \psi, WDT, PD$ 
9:    $L_{recon} \leftarrow$  Reconstruction loss using  $\Lambda(\mathbb{F}_{gt})$  and  $\Lambda(\bar{\mathbb{F}})$ 
10:  Update  $\mathbb{P}, \mathbb{W}, \psi$  to minimize  $L_{recon}$ 
11:  Apply bounds to  $\mathbb{P}$ 
12: end while
13:  $M \leftarrow$  Final mesh extracted from DMesh
14: return  $M$ 

```

Point Initialization At the start of optimization, we initialize the point positions ($\mathbb{P}$), weights ($\mathbb{W}$), and real values (ψ) using the given ground truth information ($\mathbb{P}_{gt}, \mathbb{F}_{gt}$). To be specific, we initialize the point attributes as follows.

$$\mathbb{P} = \mathbb{P}_{gt}, \quad \mathbb{W} = [1, \dots, 1], \quad \psi = [1, \dots, 1].$$

The length of vector $\mathbb{W}$ and ψ is equal to the number of points. In Figure E.4, we illustrate the initialized DMesh using these point attributes, which becomes the convex hull of the ground truth mesh.

Note that during optimization, we allow only small perturbations to the positions of initial points,

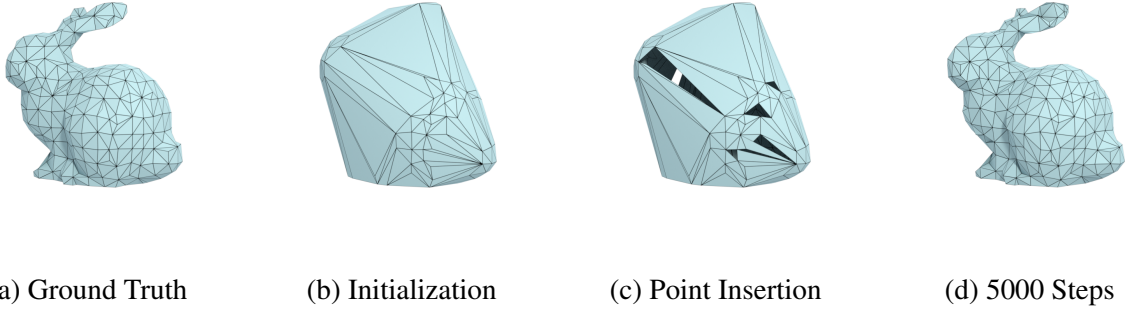


Figure E.4: **Intermediate results in converting bunny model to DMesh.** For given ground truth mesh in (a), we initialize our point attributes using the mesh vertices. (b) Then, the initial mesh becomes convex hull of the original mesh. (c) To remove undesirable faces that were not in the original mesh, we insert additional points on the undesirable faces. Then, some of them disappear because of the inserted points. (d) After optimizing 5000 steps, just before another point insertion, DMesh recovers most of the ground truth connectivity.

and fix weights and real values of them to 1. This is because we already know that these points correspond to the ground truth mesh vertices, and thus should be included in the final mesh without much positional difference. In our experiments, we set the perturbation bound as 1% of the model size.

However, we notice that we cannot restore the mesh connectivity with only small perturbations to the initial point positions, if there are no additional points that can aid the process. Therefore, we periodically perform point insertion to add additional points, which is described below.

Point Insertion The point insertion is a subroutine to add additional points to the current point configurations. It is performed periodically, at every fixed step. The additional points are placed at the random place on the faces in $\bar{\mathbb{F}}$, which correspond to the faces that should not exist in the final mesh. Therefore, these additional points can aid removing these undesirable faces.

However, we found out that inserting a point for every face in $\bar{\mathbb{F}}$ can be quite expensive. Therefore, we use k -means clustering algorithm to aggregate them into $0.1 \cdot N_F$ clusters, where N_F is the number of faces in $\bar{\mathbb{F}}$, to add the centroids of the clusters to our running point set. On top of that, we

select 1000 random faces in $\bar{\mathbb{F}}$ to put additional points directly on them. This is because there are cases where centroids are not placed on the good positions where they can remove the undesirable faces.

In Figure E.4, we render DMesh after point insertion to the initialized mesh. Note that some of the undesirable faces disappear because of the added points.

Maintaining $\bar{\mathbb{F}}$ In this problem, we minimize the reconstruction loss specified in Eq. E.8 to restore the connectivity in the ground truth mesh, and remove faces that do not exist in it. In the formulation, we denoted the faces that are comprised of mesh vertices $\mathbb{P}$, but are not included in the original mesh as $\bar{\mathbb{F}}$. Even though we can enumerate all of them, the total number of faces in $\bar{\mathbb{F}}$ mounts to $O(N^3)$, where N is the number of mesh vertices. Therefore, rather than evaluating all of those cases, we maintain a set of faces $\bar{\mathbb{F}}$ that we should exclude in our mesh during optimization.

To be specific, at each iteration, we find faces in the current WDT that are comprised of points in $\mathbb{P}$, but do not exist in $\mathbb{F}$, and add them to the running set of faces $\bar{\mathbb{F}}$. On top of that, at every pre-defined number of iterations, in our case 10 steps, we compute k -nearest neighboring points for each point in $\mathbb{P}$. Then, we find faces that can be generated by combining each point with 2 of its k -nearest points, following [13]. Then, we add the face combinations that do not belong to $\mathbb{F}$ to $\bar{\mathbb{F}}$. In our experiments, we set $k = 8$.

E.4.2 Point cloud & Multi-view Reconstruction

In Algorithm 4, we describe the overall algorithm that is used for point cloud and multi-view reconstruction tasks. We explain each step in detail below.

Algorithm 4 Point Cloud and Multi-view Reconstruction

```
1:  $T \leftarrow$  Observation (point cloud, multi-view images)
2:  $\mathbb{P}, \mathbb{W}, \psi \leftarrow$  Initialize point attributes for DMesh (using  $T$  if available)
3:  $\mathbb{F} \leftarrow \emptyset$ 
4: while epoch not ended do
5:   Initialize  $\mathbb{P}, \mathbb{W}, \psi$  with sampled points from current DMesh (if not first epoch)
6:   while step not ended do
7:      $WDT, PD \leftarrow$  Run WDT with  $\mathbb{P}, \mathbb{W}$ 
8:      $\mathbb{F} \leftarrow$  Update faces from  $WDT$ 
9:      $\Lambda(\mathbb{F}) \leftarrow$  Compute face existence probability using  $\mathbb{P}, \psi, WDT, PD$ 
10:     $L_{\text{recon}} \leftarrow$  Reconstruction loss using  $\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), T$ 
11:     $L_{\text{weight}} \leftarrow$  Weight regularization using  $PD$ 
12:     $L_{\text{real}} \leftarrow$  Realizability regularization using  $\mathbb{P}, \psi, WDT$ 
13:     $L_{\text{qual}} \leftarrow$  Quality regularization using  $\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F})$ 
14:     $L \leftarrow L_{\text{recon}} + \lambda_{\text{weight}} L_{\text{weight}} + \lambda_{\text{real}} L_{\text{real}} + \lambda_{\text{qual}} L_{\text{qual}}$ 
15:    Update  $\mathbb{P}, \mathbb{W}, \psi$  to minimize  $L$ 
16:   end while
17:    $WDT, PD \leftarrow$  Run WDT with  $\mathbb{P}, \mathbb{W}$ 
18:    $\mathbb{F} \leftarrow$  Faces in  $WDT$ 
19:    $\Lambda_{\text{wdt}}(\mathbb{F}) \leftarrow 1$ 
20:   while step not ended do
21:      $\Lambda(\mathbb{F}) \leftarrow$  Compute existence probability using  $\mathbb{P}, \psi, \Lambda_{\text{wdt}}(\mathbb{F})$ 
22:      $L_{\text{recon}} \leftarrow$  Reconstruction loss using  $\mathbb{P}, \mathbb{F}, \Lambda(\mathbb{F}), T$ 
23:      $L_{\text{real}} \leftarrow$  Realizability regularization using  $\mathbb{P}, \psi, WDT$ 
24:      $L \leftarrow L_{\text{recon}} + \lambda_{\text{real}} L_{\text{real}}$ 
25:     Update  $\psi$  to minimize  $L$ 
26:   end while
27: end while
28:  $M \leftarrow$  Final mesh from DMesh after post-processing
29: return  $M$ 
```

▷ Phase 1: Joint optimization

▷ Phase 2: Probability refinement

Two Phase Optimization We divide each optimization epoch in two phases. In the first phase (phase 1), we optimize all of the point attributes – positions, weights, and real values. However, in the second phase (phase 2), we fix the point positions and weights, and only optimize the real values.

This design aims at removing ambiguity in our differentiable formulation. That is, even though we desire face existence probabilities to converge to either 0 and 1, those probabilities can converge to the values in between. To alleviate this ambiguity, after the first phase ends, we fix the tessellation to make Λ_{wdt} for each face in $\mathbb{F}$ to either 0 or 1. Therefore, in the second phase, we only care about the faces that exist in current WDT , which have Λ_{wdt} value of 1. Then, we can only care about real values.

Note that the two differentiable renderers that we introduced in Appendix E.3.3 are designed to serve for these two phases, respectively.

Point Initialization with Sample Points In this work, we propose two point initialization methods. The first initialization method can be used when we have sample points near the target geometry in hand.

This initialization method is based on an observation that the vertices of Voronoi diagram of a point set tend to lie on the medial axis of the target geometry [268, 269]. Therefore, for the given sample point set $\mathbb{P}_{sample}$, we first build Voronoi diagram of it, and find Voronoi vertices $\mathbb{P}_{voronoi}$. Then, we merge them to initialize our point set $\mathbb{P}$:

$$\mathbb{P} = \mathbb{P}_{sample} \cup \mathbb{P}_{voronoi},$$

all of which weights are initialized to 1. Then, we set the real values (ψ) of points in $\mathbb{P}_{sample}$ as 1,

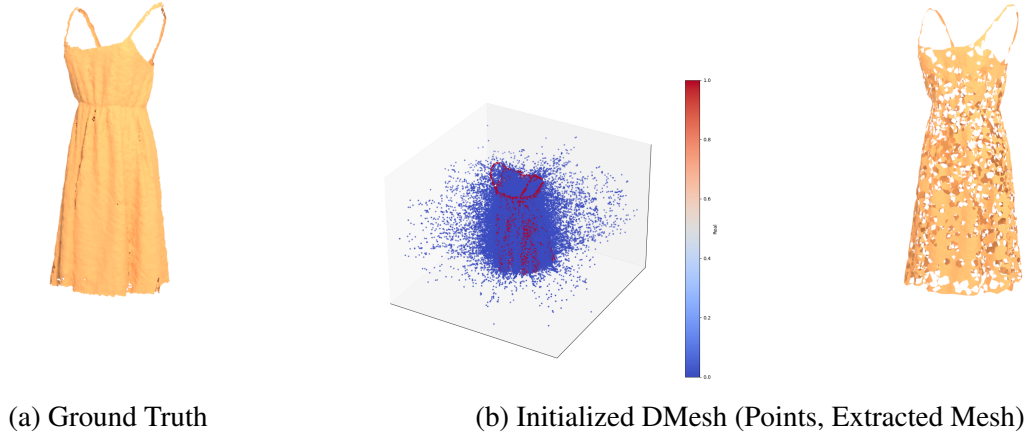


Figure E.5: **Initialized DMesh using sample points from ground truth mesh.** (a) From ground truth mesh, we uniformly sample $10K$ points to initialize DMesh. (b) In the left figure, sample points from the ground truth mesh ($\mathbb{P}_{sample}$) are rendered in red. The points that correspond to $\mathbb{P}_{voronoi}$ are rendered in blue. In the right figure, we render the initial mesh we can get from the points, which has a lot of holes.

while setting those of points in $\mathbb{P}_{voronoi}$ as 0.

In Figure E.5, we render the mesh that we can get from this initialization method, when we use $10K$ sample points. Note that the initial mesh has a lot of holes, because there could be Voronoi vertices that are located near the mesh surface, as pointed out by [269]. However, we can converge to the target mesh faster than the initialization method that we discuss below, because most of the points that we need are already located near the target geometry.

Point Initialization without Sample Points If there is no sample point that we can use to initialize our points, we initialize our points with N^3 points regularly distributed on a grid structure that encompasses the domain, all of which has weight 1 and ψ value of 1. We set $N = 20$ for every experiment (Figure E.6a). Then, we optimize the mesh to retrieve a coarse form of the target geometry (Figure E.6b). Note that we need to refine this mesh in the subsequent epochs, as explained below.

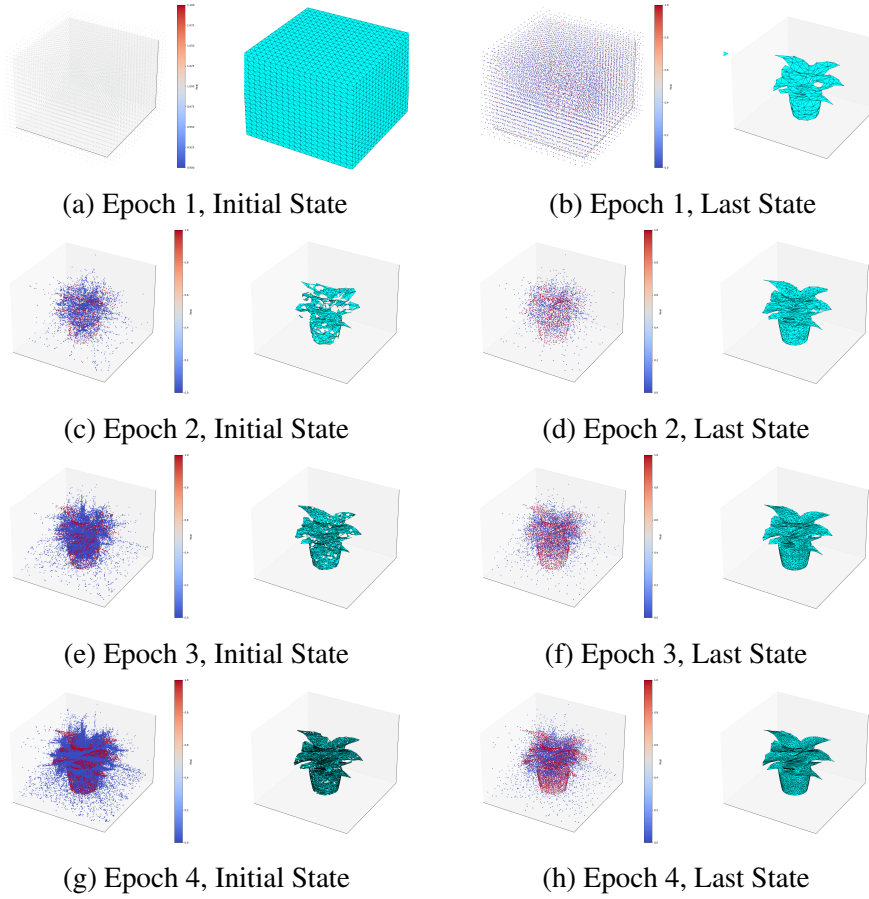


Figure E.6: **Optimization process for multi-view reconstruction for Plant model.** At each row, we present the initial state (left) and the last state (right) of each epoch. For each figure, the left rendering shows the point attributes color coded based on real values, while the right one shows the extracted mesh. (a), (b) In the first epoch, we initialize DMesh without sample points. At the end of each epoch, we sample points from the current mesh, and use them for initialization in the next epoch.

Point Initialization for Different Inputs Until now, we introduced two point initialization techniques. When the input is a point cloud, we sample subset of the point cloud to initialize our mesh (Figure E.5). However, when the input is multi-view images, we start from initialization without sample points (Figure E.6), because there is no sample point cloud that we can make use of.

Maintaining $\mathbb{F}$ We maintain the running set of faces to evaluate probability existence for in $\mathbb{F}$. At each iteration, after we get WDT , we insert every face in WDT to $\mathbb{F}$, as it has a high possibility

to persist in the subsequent optimization steps. Also, as we did in mesh to DMesh conversion (Appendix E.4.1), at every 10 optimization step, we find k -nearest neighbors for each point, and form face combinations based on them. Then, we add them to $\mathbb{F}$.

Mesh Refinement At start of each epoch, if it is not the first epoch, we refine our mesh by increasing the number of points. To elaborate, we refine our mesh by sampling N number of points on the current DMesh, and then initialize point attributes using those sample points as we explained above. We increase N as number of epoch increases. For instance, in our multi-view reconstruction experiments, we set the number of epochs as 4, and set $N = (1K, 3K, 10K)$ for the epochs excluding the first one. In Figure E.6, we render the initial and the last state of DMesh of each epoch. Note that the mesh complexity increases and becomes more accurate as epoch proceeds, because we use more points. Therefore, this approach can be regarded as a coarse-to-fine approach.

E.4.3 Post-Processing

When it comes to multi-view reconstruction, we found out that it is helpful to add one more constraint in defining the face existence. In our formulation, in general, a face F has two tetrahedra (T_1, T_2) that are adjacent to each other over the face. Then, we call the remaining point of T_1 and T_2 that is not included in F as P_1 and P_2 . Our new constraint requires at least one of P_1 and P_2 to have ψ value of 0 to let F exist.

This additional constraint was inspired by the fact that F is not visible from outside if F exists in our original formulation, and both of P_1 and P_2 have ψ value of 1. That is, if it is not visible from outside, we do not recognize its existence. This constraint was also adopted to accommodate our real regularization, which increases the real value of points near surface. If this regularization makes the

real value of points inside the closed surface, they would end up in internal faces that are invisible from outside. Because of this invisibility, our loss function cannot generate a signal to remove them. In the end, we can expect all of the faces inside a closed surface will exist, because of the absence of signal to remove them. Therefore, we choose to remove those internal faces by applying this new constraint in the post-processing step.

Note that this discussion is based on the assumption that our renderer does a precise depth testing. If it does not do the accurate depth testing, internal faces can be regarded as visible from outside, and thus get false gradient signal. In Figure E.3a, the final mesh after phase 1 is rendered, and we can see there are lots of internal faces as the renderer used in phase 1 does not support precise depth testing. However, we can remove them with the other renderer in phase 2, as shown in Figure E.3b, which justifies our implementation of two different renderers.

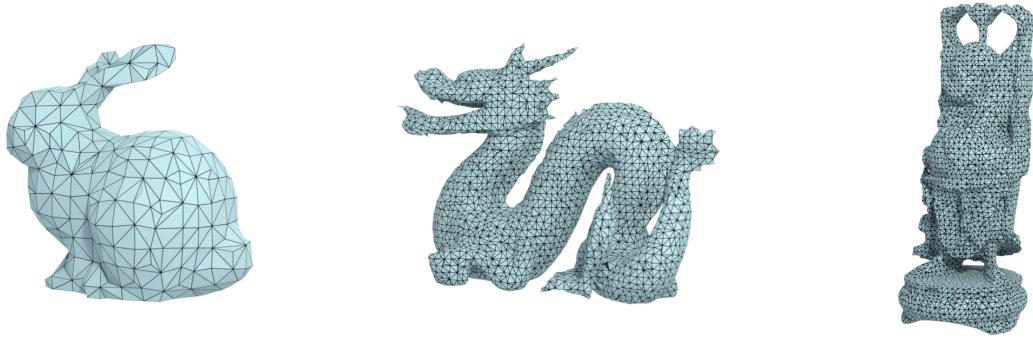
Finally, we note that this constraint is not necessary for point cloud reconstruction, because if we minimize CD_{ours} in Appendix E.3.2, the internal faces will be removed automatically.

E.5 Experimental Details

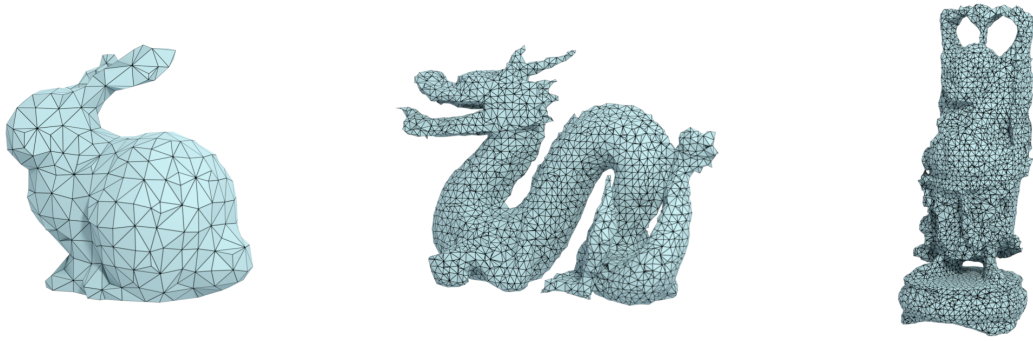
In this section, we provide experimental details for the results in Section 7.5, and visual renderings of the our reconstructed mesh. Additionally, we provide the results of ablation studies about regularizations that we suggested in Section 8.3.4.

E.5.1 Mesh to DMesh

As shown in Table 6.1, we reconstruct the ground truth connectivity of Bunny, Dragon, and Buddha model from Stanford dataset [270]. For all these experiments, we optimized for $20K$ steps, and



(a) Ground Truth Mesh



(b) Reconstructed DMesh

Figure E.7: **Reconstruction results for mesh to DMesh experiment. From Left: Bunny, Dragon, and Buddha.** We can observe that most of the edge connectivity is perserved in the reconstruction, even though the appearance is slightly different from the ground truth mesh because of small perturbations of vertex positions.

used an ADAM optimizer [260] with learning rate of 10^{-4} . For Bunny model, we inserted additional points at every 5000 step. For the other models, we inserted them at every 2000 step.

In Figure E.7, we provide the ground truth mesh and our reconstructed mesh. We can observe that most of the connectivity is preserved in our reconstruction, as suggested numerically in Table 6.1. However, note that the appearance of the reconstructed mesh can be slightly different from the ground truth mesh, because we allow 1% of positional perturbations to the mesh vertices.

E.5.2 Point Cloud & Multi-view Reconstruction

E.5.2.1 Hyperparameters for Point Cloud Reconstruction

- Optimizer: ADAM Optimizer, Learning rate = 10^{-4} for open surface meshes and two mixed surface meshes (Bigvegas, Raspberry) / $3 \cdot 10^{-4}$ for closed surface meshes, and one mixed surface mesh (Plant).
- Regularization: $\lambda_{weight} = 10^{-8}$, $\lambda_{real} = 10^{-3}$, $\lambda_{qual} = 10^{-3}$ for every mesh.
- Number of epochs: Single epoch for every mesh.
- Number of steps per epoch: 1000 steps for phase 1, 500 steps for phase 2 for every mesh.

E.5.2.2 Hyperparameters for Multi-view Reconstruction

- Optimizer: ADAM Optimizer, Learning rate = 10^{-3} in the first epoch, and $3 \cdot 10^{-4}$ in the other epochs for every mesh.
- Weight Regularization: $\lambda_{weight} = 10^{-8}$ for every mesh.
- Real Regularization: $\lambda_{real} = 10^{-3}$ for the first 100 steps in every epoch for open surface meshes and one mixed surface mesh (Plant) / 10^{-2} for the first 100 steps in every epoch for closed surface meshes and two mixed surface meshes (Bigvegas, Raspberry).
- Quality Regularization: $\lambda_{qual} = 10^{-3}$ for every mesh.
- Normal Coefficient: $\lambda_{normal} = 0$ for every mesh (Eq. [E.10](#)).

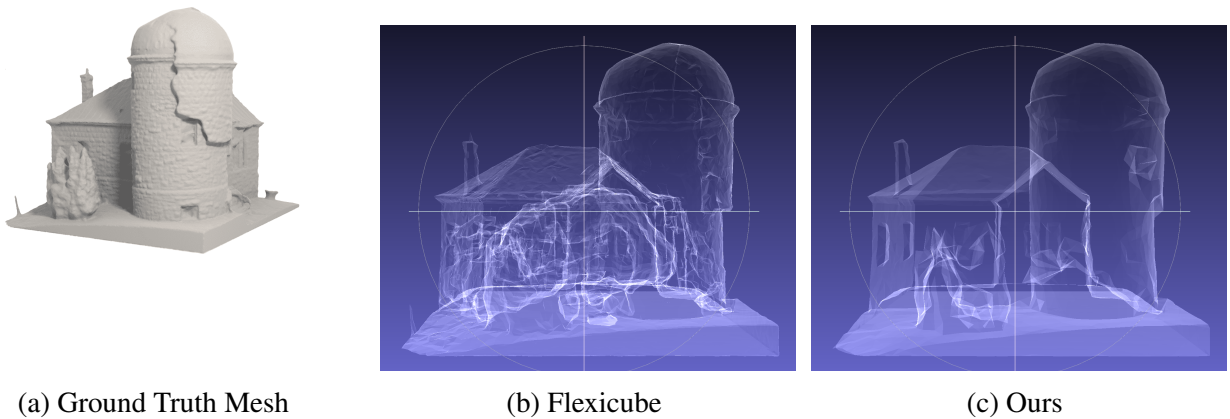


Figure E.8: **Reconstruction results for a closed surface model in Thingi32 dataset.** Flexicube [11] can generate internal structures, while our approach removes them through post-processing.

- Number of epochs: 4 epochs for every mesh. In the first epoch, use 20^{-3} regularly distributed points for initialization. In the subsequent epochs, sample $1K$, $3K$, and $10K$ points from the current mesh for initialization.
- Number of steps per epoch: 500 steps for phase 1, 500 steps for phase 2 for every mesh.
- Batch size: 64 for open surface meshes, 16 for the other meshes.

E.5.2.3 Visual Renderings

In Figure E.12, E.13, and E.14, we provide visual renderings of our point cloud and multi-view reconstruction results with ground truth mesh. We also provide illustration of input point cloud and diffuse map. Note that we also used depth renderings for multi-view reconstruction experiments.

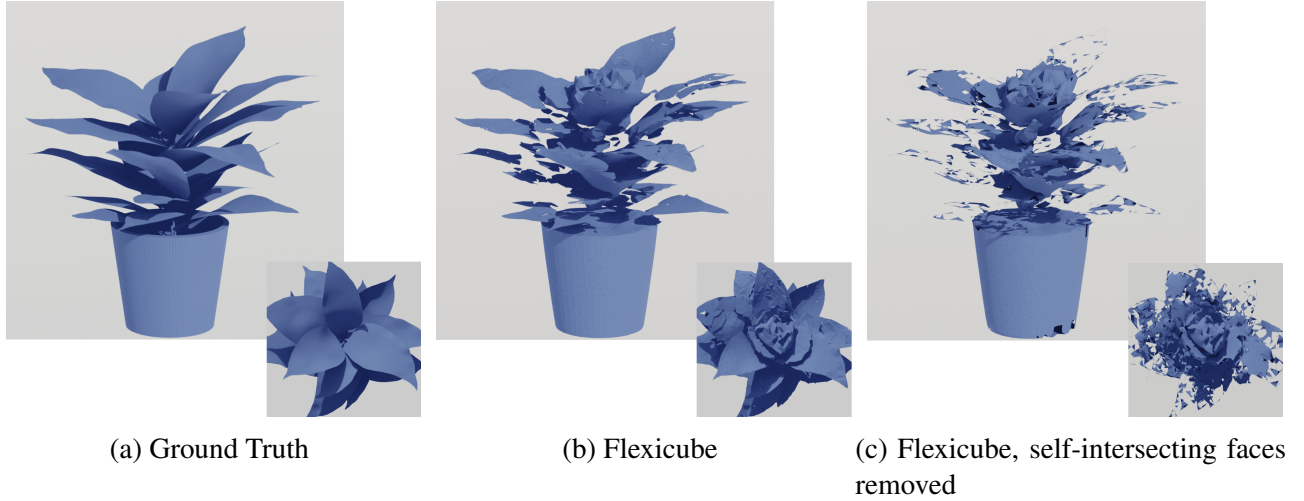


Figure E.9: **Reconstruction results for the Plant model.** Flexicube [11] can generate redundant, self-intersecting faces for open surfaces, in this case, leaves. To better capture the redundant faces, we rendered the models from upper side, which is shown in the bottom right figures.

E.5.2.4 Additional Discussion

Generally, we can observe that reconstruction results from both point cloud and multi-view images capture the overall topology well. However, we noticed that the multi-view reconstruction results are not as good as point cloud reconstruction results. In particular, we can observe small holes in the multi-view reconstruction results. We assume that these artifacts are coming from relatively weaker supervision of multi-view images than dense point clouds. Also, we believe that we can improve these multi-view reconstruction results with more advanced differentiable renderer, and better mesh refinement strategy. In the current implementation, we lose connectivity information at the start of each epoch, which is undesirable. We believe that we can improve this approach by inserting points near the regions of interest, rather than resampling over entire mesh.

Also, regarding comparison to Flexicube [11] in Table 6.2, we tried to found out the reason why ours give better results than Flexicube in terms of CD to the ground truth mesh for closed surfaces in

thingi32 dataset. We could observe that Flexicube’s reconstruction results capture fine geometric details on the surface mesh, but also observed that they have lots of false internal structure (Figure E.8). Note that this observation not only applies to closed surfaces, but also to open surfaces, where it generates lots of false, self-intersecting faces (Figure E.9). Our results do not suffer from these problems, as we do post-processing (Appendix E.4.2) to remove inner structure, and also our method can represent open surfaces better than the volumetric approaches without self-intersecting faces.

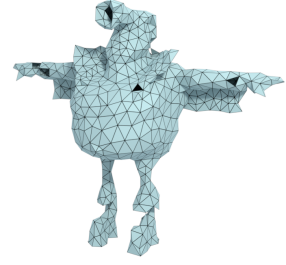
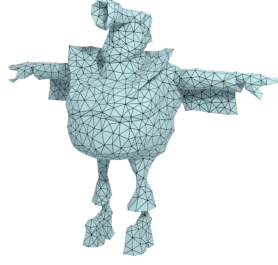
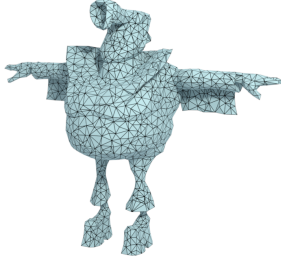
E.5.3 Ablation studies

In this section, we provide ablation studies for the regularizations that we proposed in Section 8.3.4. We tested the effect of the regularizations on the point cloud reconstruction task.

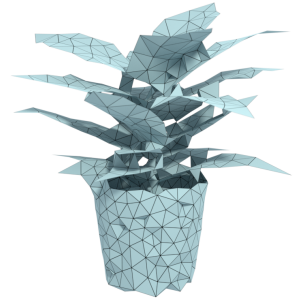
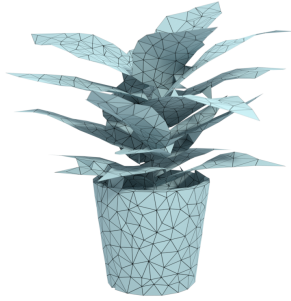
E.5.3.1 Weight Regularization

We tested the influence of weight regularization in the final mesh, by choosing λ_{weight} in $(10^{-6}, 10^{-5}, 10^{-4})$. Note that we set the other experimental settings as same as described in Section E.5.2, except $\lambda_{quality}$, which is set as 0, to exclude it from optimization.

In Table E.3, we provide the quantitative results for the experiments. For different λ_{weight} , we reconstructed mesh from point clouds, and computed average Chamfer Distance (CD) and average number of faces across every test data. We can observe that there exists a clear tradeoff between CD and mesh complexity. To be specific, when $\lambda_{weight} = 10^{-6}$, the CD is not very different from the results in Table 6.2, where we use $\lambda_{weight} = 10^{-8}$. However, when it increases to 10^{-5} and 10^{-4} , we can observe that the mesh complexity (in terms of number of faces) decreases, but CD increases quickly.



(a) Bigvegas



(b) Plant

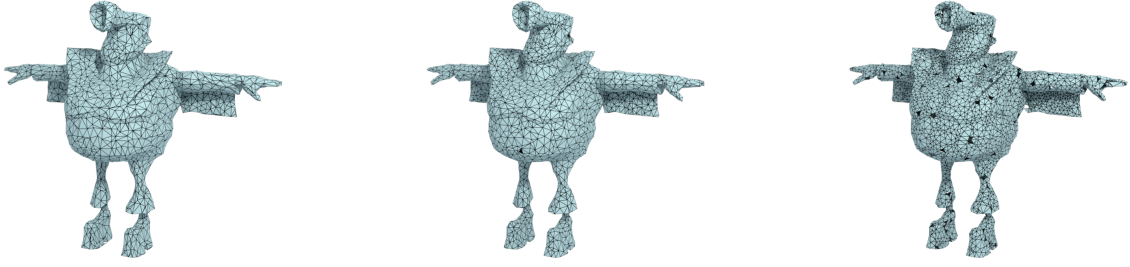
Figure E.10: **Point cloud reconstruction results with different λ_{weight} .** From Left: $\lambda_{weight} = 10^{-6}, 10^{-5}$, and 10^{-4} .

The renderings in Figure E.10 support these quantitative results. When $\lambda_{weight} = 10^{-6}$, we can observe good reconstruction quality. When $\lambda_{weight} = 10^{-5}$, there are small artifacts in the reconstruction, but we can get meshes of generally good quality with fewer number of faces. However, when it becomes 10^{-4} , the reconstruction results deteriorate, making holes and bumpy faces on the smooth surface.

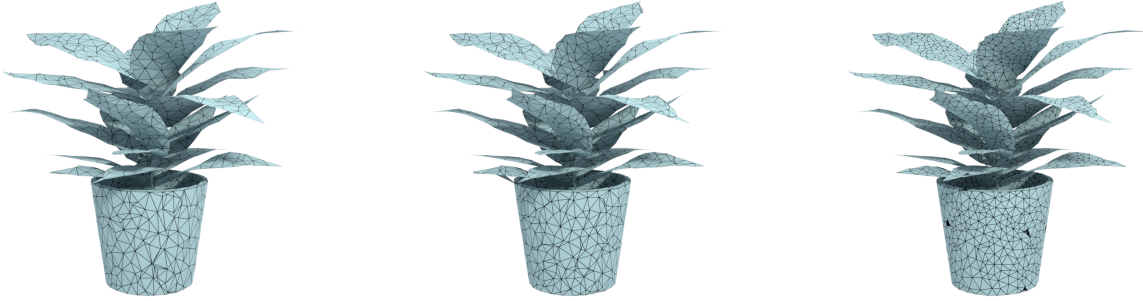
Therefore, we can conclude that weight regularization contributes to reducing the mesh complexity. However, we need to choose λ_{weight} care-

Table E.3: Ablation study for weight regularization, quantitative results.

λ_{weight}	10^{-6}	10^{-5}	10^{-4}
CD	7.48	8.08	10.82
Num. Face	4753	2809	1786



(a) Bigvegas



(b) Plant

Figure E.11: **Point cloud reconstruction results with different $\lambda_{quality}$.** From Left: $\lambda_{real} = 10^{-4}, 10^{-3}$, and 10^{-2} .

fully, so that it does not harm the reconstruction quality. The experimental results tell us setting λ_{weight} to 10^{-6} could be a good choice to balance between these two contradictory objectives.

E.5.3.2 Quality Regularization

As we did in the previous section, we test the influence of quality regularization in the final mesh by selecting λ_{real} among $(10^{-4}, 10^{-3}, 10^{-2})$. We also set the other experimental settings as same as before, except $\lambda_{weight} = 0$.

In Table E.4 and Figure E.11, we present quantitative and qualitative comparisons between the reconstruction results. We provide statistics about average CD, average number of faces, and average aspect ratio of faces. Interestingly, unlike weight regularization, we could not observe tradeoff between CD and aspect ratio. Rather than that, we could find that CD decreases as aspect ratio gets smaller, and thus the triangle quality gets better.

Table E.4: Ablation study for quality regularization, quantitative results.

λ_{qual}	10^{-4}	10^{-3}	10^{-2}
CD	7.60	7.42	7.28
Num. Face	8266	8349	10806
Aspect Ratio	2.33	2.06	1.55

We find the reason for this phenomenon in the increase of smaller, good quality triangle faces. Note that there is no significant difference between the number of faces between $\lambda_{qual} = 10^{-4}$ and 10^{-3} . Also, we cannot find big difference between visual renderings between them, even though the aspect ratio was clearly improved. However, when λ_{qual} becomes 10^{-2} , the number of faces increase fast, which can be observed in the renderings, too. We believe that this increase stems from our quality constraint, because it has to generate more triangles to represent the same area, if there is less degree of freedom to change the triangle shape. Since it has more triangle faces, we assume that they contribute to capturing fine details better, leading to the improved CD.

However, at the same time, note that the number of holes increase as we increase λ_{qual} , which lead to visual artifacts. We assume that there are not enough points to remove these holes, by generating quality triangle faces that meet our needs. Therefore, as discussed before, if we can find a systematic way to prevent holes, or come up with a better optimization scheme to remove them, we expect that we would be able to get accurate mesh comprised of better quality triangles.

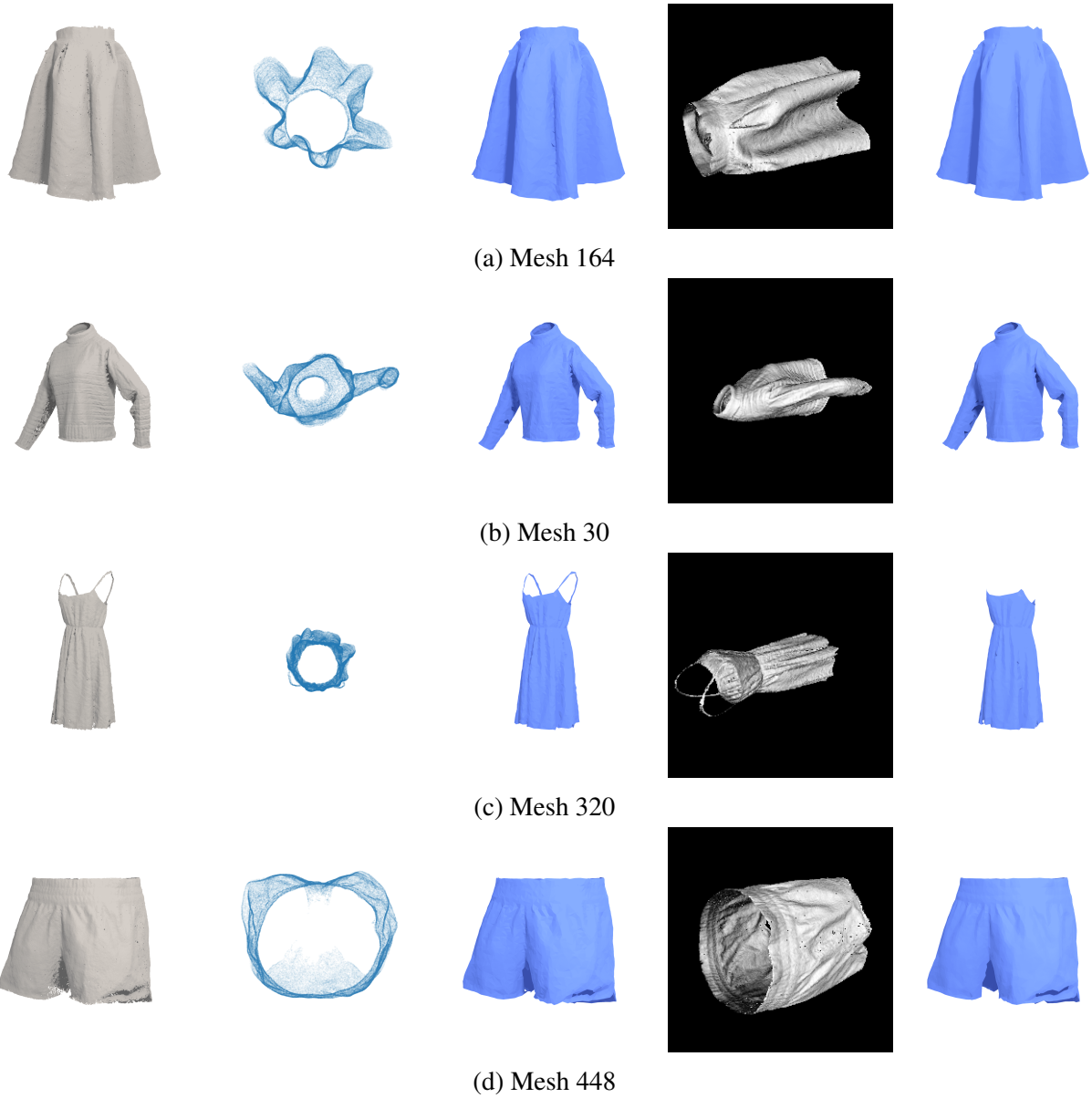


Figure E.12: **Point cloud and Multi-view Reconstruction results for open surface models.** From Left: Ground truth mesh, sample point cloud, point cloud reconstruction results, diffuse rendering, multi-view reconstruction results.

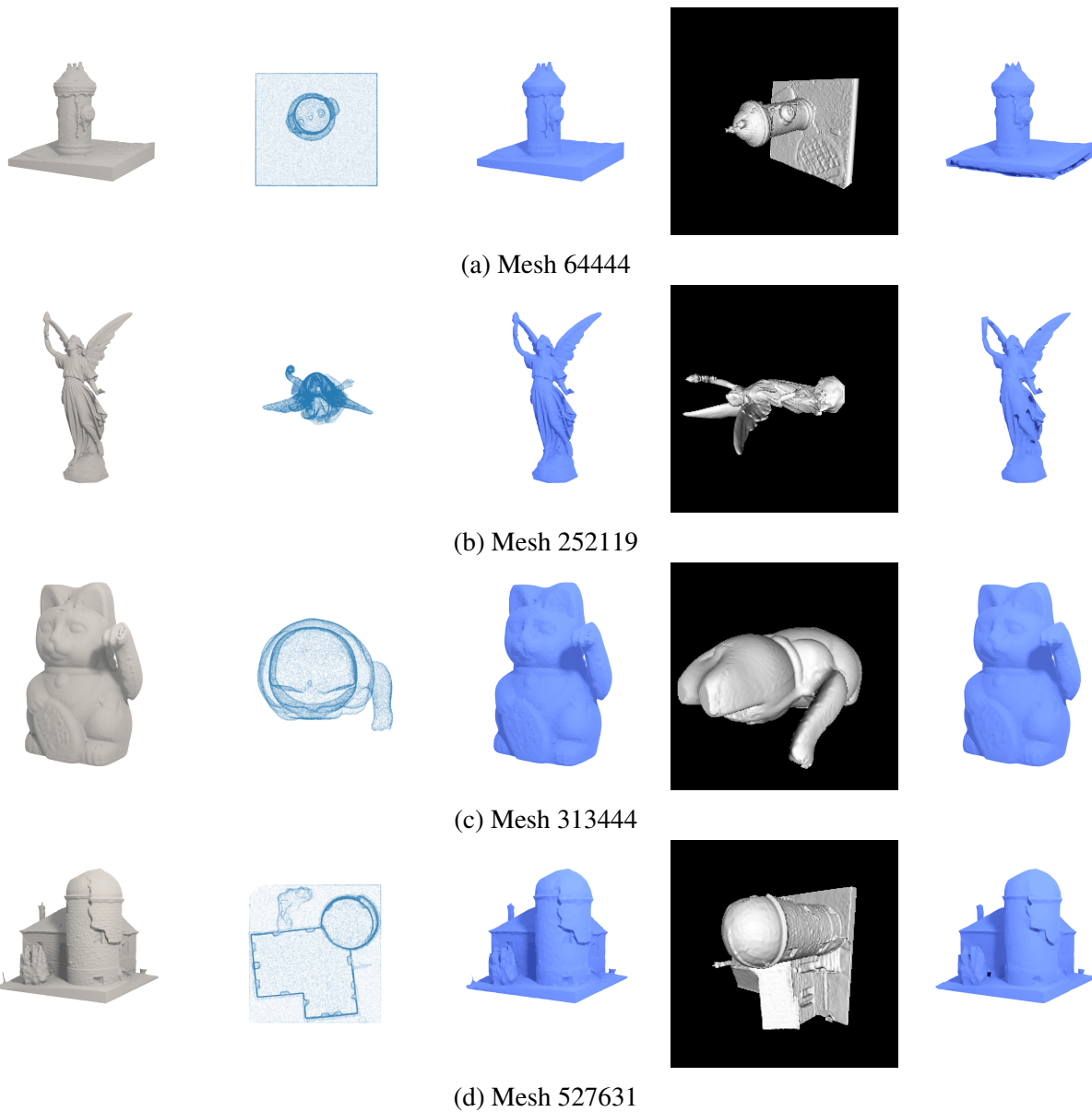


Figure E.13: **Point cloud and Multi-view Reconstruction results for closed surface models.** From Left: Ground truth mesh, sample point cloud, point cloud reconstruction results, diffuse rendering, multi-view reconstruction results.

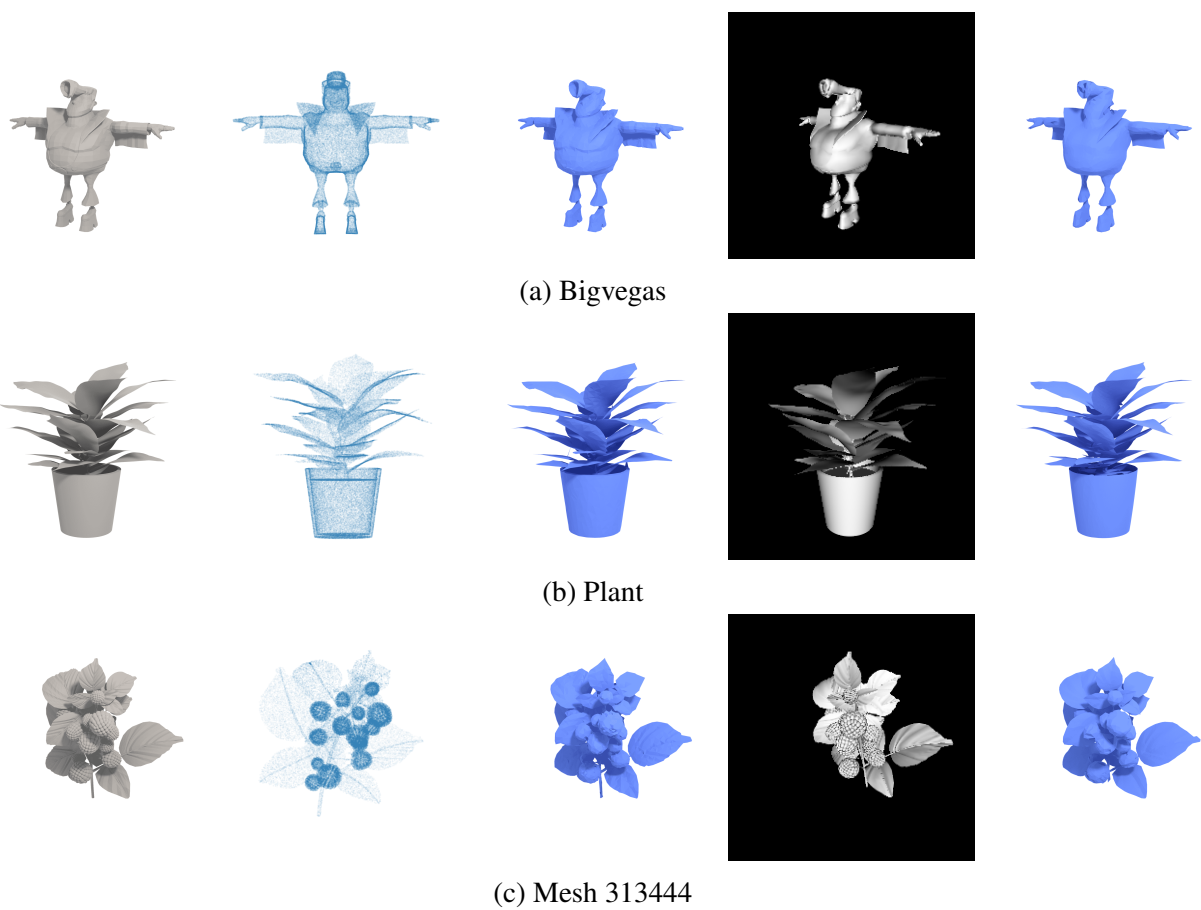


Figure E.14: **Point cloud and Multi-view Reconstruction results for mixed surface models.** From Left: Ground truth mesh, sample point cloud, point cloud reconstruction results, diffuse rendering, multi-view reconstruction results.

Appendix F: DMesh++: An Efficient Differentiable Mesh for Complex Shapes

F.1 Details about *Minimum-Ball* algorithm

F.1.1 Algorithm

Algorithm 5 Minimum-Ball

```

1:  $\mathbb{P}, \mathbb{F} \leftarrow$  Set of points and query faces
2:  $\alpha_{min} \leftarrow$  Coefficient for sigmoid function
3:  $B_{\mathbb{F}}^c, B_{\mathbb{F}}^r \leftarrow$  Compute-Minimum-Ball( $\mathbb{P}, \mathbb{F}$ )
4:  $P_{\mathbb{F}}^{nearest} \leftarrow$  Find-Nearest-Neighbor( $B_{\mathbb{F}}^c, \mathbb{P}$ )
5:  $d(B_{\mathbb{F}}, \mathbb{P}) \leftarrow B_{\mathbb{F}}^r - ||P_{\mathbb{F}}^{nearest} - B_{\mathbb{F}}^c||$ 
6:  $\lambda_{min}(\mathbb{F}) \leftarrow \sigma(d(B_{\mathbb{F}}, \mathbb{P}) \cdot \alpha_{min})$ 
7: return  $\lambda_{min}(\mathbb{F})$ 

```

We formally describe the *Minimum-Ball algorithm* in [algorithm 5](#).

- **Line 1:** We define the given set of points (specifically, their positions) as $\mathbb{P}$ and the query faces as $\mathbb{F}$.
- **Line 2:** We introduce α_{min} , the coefficient for the sigmoid function used to map the signed distance to a probability. Details on determining α_{min} are provided in [Appendix F.1.3](#).
- **Line 3:** For each query face $F \in \mathbb{F}$, we compute the minimum bounding ball (B_F) as described in [Appendix F.1.2](#). We denote the entire set of bounding balls as $B_{\mathbb{F}}$, their centers as $B_{\mathbb{F}}^c$, and their radii as $B_{\mathbb{F}}^r$.
- **Line 4:** For each $F \in \mathbb{F}$, we find the nearest neighbor of B_F^c in $\mathbb{P} - F$. However, this operation cannot be parallelized across all query faces because the set $\mathbb{P} - F$ varies for each face. To

address this, we find $(d + 1)$ -nearest neighbors of B_F^c in $\mathbb{P}$, where d is the spatial dimension.

This approach ensures correctness in two scenarios:

- If $F \in \mathbb{F}_{min}$, the bounding ball B_F does not contain any points from $\mathbb{P}$ within it, and the points on F are the d -nearest neighbors of B_F^c . To find the nearest neighbor in $\mathbb{P} - F$, we need to consider $(d + 1)$ -nearest neighbors.
- If $F \notin \mathbb{F}_{min}$, only the single nearest neighbor of B_F^c is relevant.

To safely handle both cases, we always search for $(d + 1)$ -nearest neighbors and then select the first neighbor from the list that does not belong to F .

- **Line 5:** We compute the signed distance $d(B_F, \mathbb{P})$ for all query faces.
- **Line 6:** The signed distance is converted to a probability using the sigmoid function, with α_{min} as the scaling factor.
- **Line 7:** Finally, the algorithm returns the computed probabilities for all faces.

F.1.2 Minimum-Ball computation

Let us define a face $F = \{p_1, p_2, \dots, p_d\}$, where $p_i \in \mathbb{P}$. To determine the bounding balls of F , we first identify the set of points that are equidistant from the vertices of F . Among these, we select the point lying on the hyperplane containing F as the center of the minimum bounding ball, denoted as B_F^c .

When $d = 2$, the center simplifies to the midpoint of F :

$$B_F^c|_{d=2} = \frac{1}{2}(p_1 + p_2). \quad (\text{F.1})$$

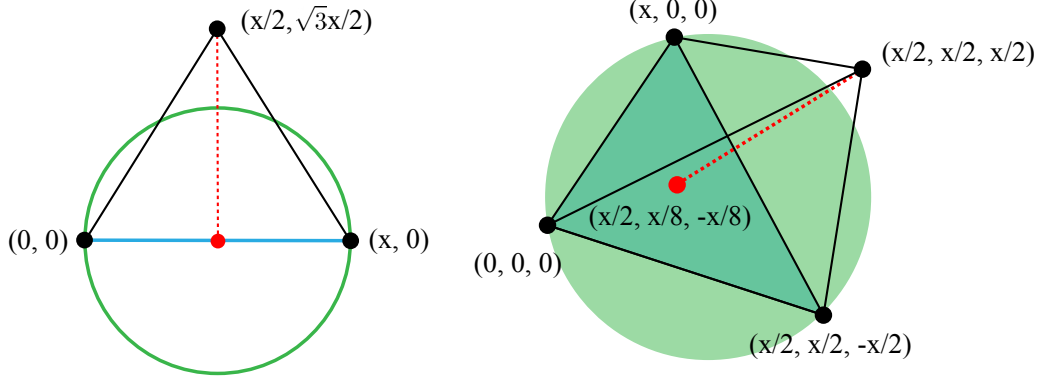


Figure F.1: **Common signed distance for a 2D (left) and 3D (right) face in (initial) regular grid.** We compute the signed distance by subtracting the radius of the **minimum bounding ball** from the length of the **red line**. The **red dot** represents the center of the **minimum bounding ball**.

For $d = 3$, the computation is more complex ¹:

$$B_F^c|_{d=3} = p_1 + \frac{\|d_2\|^2(d_1 \times d_2) \times d_1 + \|d_1\|^2(d_2 \times d_1) \times d_2}{2\|d_1 \times d_2\|^2}, \quad (\text{F.2})$$

where $d_1 = p_2 - p_1$ and $d_2 = p_3 - p_1$.

Unlike the case where $d = 2$, for $d = 3$, we cannot compute B_F^c if $\|d_1 \times d_2\| = 0$. During computation, cases where this value falls below a certain threshold are marked and excluded from subsequent steps to avoid numerical instability.

After determining B_F^c , we calculate the radius B_F^r as the distance between B_F^c and the points on F .

F.1.3 Sigmoid coefficient α_{min}

The sigmoid coefficient α_{min} plays a critical role in determining the probability to which a signed distance d is mapped. Even if a face F satisfies the *Minimum-Ball* condition by a large margin

¹Derived from the Geometry Junkyard: <https://ics.uci.edu/~eppstein/junkyard/circumcenter.html>

($d(B_F, \mathbb{F}) \gg 0$), indicating a high existence probability for F , a small α_{min} value would result in the derived probability being only slightly greater than 0.5. To minimize such mismatches, we set α_{min} based on the density of the grid from which optimization begins.

As discussed in Appendix F.2.2.1, the reconstruction process often starts from a fixed triangular (2D) or tetrahedral (3D) grid (fig. F.4). At the initial state, every face in the grid satisfies the *Minimum-Ball* condition (Appendix F.2.2.1). Notably, every interior face in the grid shares a common signed distance $d_{common} > 0$. Let us denote x as the edge length of the grid, applicable for both 2D and 3D cases. Then, the common signed distance can be computed as follows:

For $d = 2$:

$$d_{common} = \frac{\sqrt{3} - 1}{2}x. \quad (\text{F.3})$$

For $d = 3$:

$$d_{common} = \frac{\sqrt{34} - 3\sqrt{2}}{8}x. \quad (\text{F.4})$$

In fig. F.1, we provide an illustration of the reasoning behind these results. By calculating these common signed distances, we use them to determine α_{min} . Specifically, during the first epoch, we set $\alpha_{min} = 32/d_{common}$, ensuring that the probability for every face in the grid is initialized to $\sigma(32) \simeq 1.0$.

In subsequent epochs, α_{min} is adjusted to account for the additional points introduced during subdivision. If α_{min}^1 represents the value in the first epoch, the value for the i -th epoch is given by:

$$\alpha_{min}^i = \frac{\alpha_{min}^1}{2^{i-1}}. \quad (\text{F.5})$$

F.1.4 Nearest neighbor caching

In [sections 7.3.2](#) and [7.5.1](#), we demonstrated how the *Minimum-Ball* algorithm significantly accelerates tessellation. This process can be further optimized by periodically caching the K -nearest neighbors of each B_F^c in $\mathbb{P}$ and using the cached neighbors for computing probabilities until the next cache update. This optimization is feasible because the K -nearest neighbors generally do not change significantly during the optimization process.

Let us define the number of optimization steps as n_0 and the number of steps between cache updates as n_1 . At every n_1 steps, we refresh the query faces $\mathbb{F}$ based on the current set of points $\mathbb{P}$ and recompute the centers of the minimum bounding balls for the query faces ($B_{\mathbb{F}}^c$). Then, we identify the K -nearest neighbors of $B_{\mathbb{F}}^c$ in $\mathbb{P}$. In practice, we compute the $(K + d)$ -nearest neighbors instead, as explained in [Appendix F.1.1](#), to ensure robustness.

During the subsequent optimization steps, for a given face F , we compute the distance from B_F^c to the cached K -nearest neighbors in $\mathbb{P}$ and select the nearest neighbor from the cache to compute the signed distance $d(B_F, \mathbb{P})$. This mechanism is described in detail in [algorithm 6](#) and [Appendix F.2.2.2](#) in the context of point position optimization. In our experiments for 3D multi-view reconstruction, we set $n_0 = 2000$, $n_1 = 50$, and $K = 10$.

F.2 Details about Reconstruction Process

In this section, we provide implementation details about our reconstruction process described in [section 7.4](#). Before delving into these details, we introduce the loss formulations for reconstruction problems.

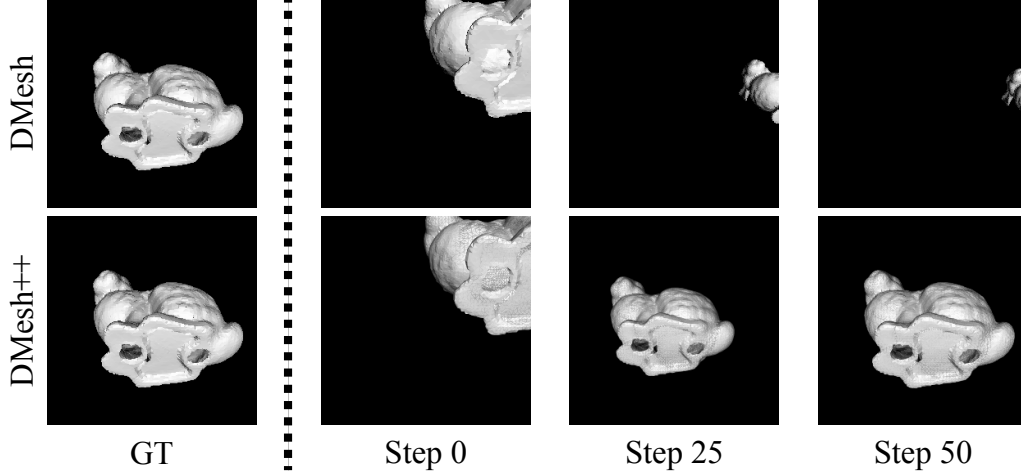


Figure F.2: **Role of visibility gradient in geometric optimization.** In this experiment, we optimize the translation vector of the object by comparing its rendered image and the ground truth image on the left. Since the differentiable renderer of DMesh [20] does not implement visibility gradient, while ours does, DMesh fails to find the correct translation vector.

F.2.1 Loss Formulation

Our final loss, L , is comprised of main reconstruction loss (L_{recon}) and two regularization terms:

L_{qual} and L_{real} .

$$L = L_{recon} + \lambda_{qual} \cdot L_{qual} + \lambda_{real} \cdot L_{real}. \quad (\text{F.6})$$

We explain each of these terms below.

F.2.1.1 Reconstruction Loss (L_{recon})

Reconstruction loss drives the reconstruction process by comparing our current probabilistic mesh and the given ground truth observations. For different observations, we need different loss functions as follows.

Point Cloud When ground truth point clouds are provided, we utilize the expected Chamfer Distance (CD) proposed by [20]. In this formulation, when sampling points from our mesh, we assign an existence probability to each sampled point, which matches the probability of the face from which the point is sampled. The expected CD incorporates these probabilities, unlike the traditional Chamfer Distance, which does not. For further details, refer to [20].

Multi-view Images For rendering probabilistic faces, we interpret each face’s existence probability as its opacity, following [20]. To render large number of semi-transparent faces efficiently, we use the differentiable renderer of [20], but we found out that it does not implement visibility gradient that is necessary for optimizing geometric properties (fig. F.2). For the details about this visibility gradient, please refer to [24], which implemented the visibility gradient using anti-aliasing. Following their path, we enhanced the differentiable renderer of DMesh by implementing anti-aliasing in CUDA, which provides us visibility gradients.

In fig. F.3, we briefly illustrate how we implemented anti-aliasing in CUDA. Specifically, for each (triangular) face-pixel pair (F, P) , we project F onto the image space and compute the overlapping area $A(F, P)$ between the projected triangle and the pixel (fig. F.3 right, blue area). Denoting the total area of the pixel as $A(P)$, the ratio of the overlapping area in the given pixel, $\rho(F, P)$, is computed as:

$$\rho(F, P) = \frac{A(F, P)}{A(P)}. \quad (\text{F.7})$$

We use $\rho(F, P)$ to determine the opacity of the face F at the pixel P . If the opacity of F is $\alpha(F)$, we compute the face opacity at pixel P , $\alpha(F, P)$, as:

$$\alpha(F, P) = \alpha(F) \cdot \rho(F, P) \leq \alpha(F). \quad (\text{F.8})$$

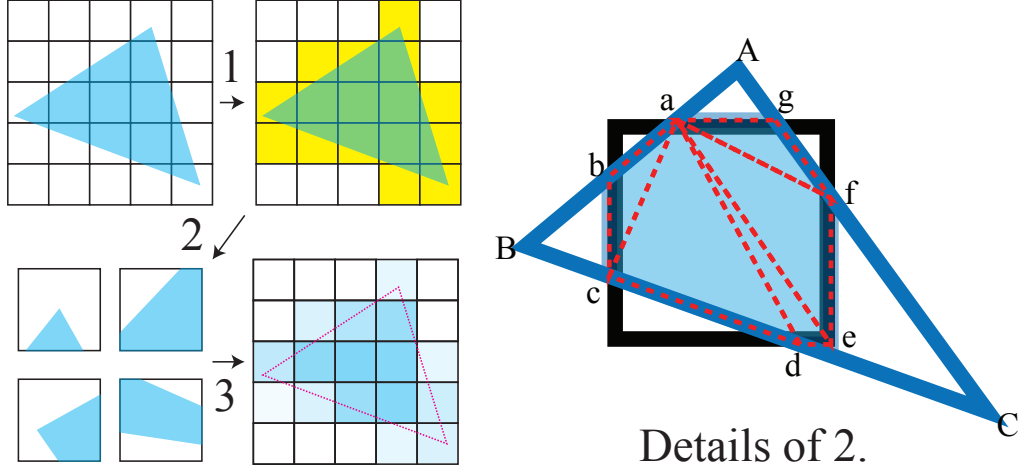


Figure F.3: **Implementation of anti-aliasing in our differentiable renderer.** On the left, we show the process of anti-aliasing: 1) Find pixels that overlap with the given triangle, 2) Find the area that each pixel overlaps with the given triangle, and 3) Determine the color of each pixel based on the overlap area. On the right, we show details of the step 2.

Thus, the opacity of F at P is proportional to the overlapping area between the triangle and the pixel.

In the right figure of [fig. F.3](#), we illustrate the process of computing the overlapping area $A(F, P)$. The vertices of F are projected onto the image plane and visited in counterclockwise order (e.g., A - B - C - A in the illustration). We then find the intersection points between the triangle edges and the pixel boundaries. These intersection points form the vertices of the (convex) overlapping polygon.

For example, vertices (a) and (b) are found by calculating the intersections of $\overline{AB}$ with the pixel boundaries. The vertices of the overlapping polygon are stored in counterclockwise order, and the polygon is subdivided into a set of sub-triangles, as shown by the dotted red lines in the visualization. The total area of the overlapping polygon is obtained by summing the areas of the sub-triangles.

Using this enhanced differentiable renderer, we render multi-view images and compute the L_1 loss between the ground truth images as the reconstruction loss, L_{recon} .

F.2.1.2 Triangle Quality Loss (L_{qual})

To improve the triangle quality of the final mesh, we adopt the triangle quality loss (L_{qual}) of DMesh [20]. Specifically, the loss is defined as follows:

$$L_{qual} = \frac{1}{|\mathbb{F}|} \sum_{F \in \mathbb{F}} AR(F) \cdot \Lambda(F), \quad (\text{F.9})$$

where $\mathbb{F}$ is the set of every face combination we consider, $AR(\cdot)$ is a function that computes aspect ratio of the given face, and $\Lambda(\cdot)$ is the face probability function defined in [section 7.3.1](#).

F.2.1.3 Real Loss (L_{real})

We minimize the sum of point-wise real values (ψ), so that we can remove redundant faces as much as possible during optimization. The loss L_{real} is simply defined as:

$$L_{real} = \frac{1}{|\mathbb{P}|} \sum_{p \in \mathbb{P}} \Psi(p), \quad (\text{F.10})$$

where $\Psi(\cdot)$ is the function that returns the real value of the given point, as defined in [section 7.3.1](#).

F.2.2 Reconstruction Steps

Now we provide detailed explanations about each step in the reconstruction process.

F.2.2.1 Step 1: Initialization

We initialize our point features differently for two different scenarios: when sample point cloud is given or not.

Point Cloud Init. When a point cloud sampled from the target shape is given, we can initialize our point features using the point cloud, so that the initial configuration already captures the overall structure of the target shape (fig. 7.3). Specifically, for the given point cloud, we estimate the density of the point cloud by computing the distance to the nearest point for each point. Then, we down sample the point cloud using a voxel grid, of which size is defined as the point cloud density, to remove redundant points and holes. Finally, we follow the initialization scheme of DMesh [20] using the down sampled point cloud.

Regular Grid Init. If we do not have any prior knowledge about the target shape, we first organize the points in a regular grid, ensuring that every face in the grid satisfies the *Minimum-Ball* condition (Definition 7.3.1), and initialize the point-wise real values (ψ) with additional features (colors). This regular grid guarantees that the faces observed in this step will also be observable in the subsequent step, where the *Minimum-Ball* algorithm determines face existence. For $d = 2$, this condition is satisfied by forming every triangle in the grid as an equilateral triangle. For $d = 3$, we use a body-centered cubic lattice. The grids are illustrated in fig. F.4.

With these fixed points and faces, we formulate the final loss by setting $\lambda_{qual} = 0$ and $\lambda_{real} = 10^{-4}$ in eq. (F.6), and minimize it to determine which faces to include in the final mesh. After optimization, we collect points with real values larger than 0.01 to ensure that as many faces as possible are available for the next optimization step, thereby reducing the risk of holes in the surface.

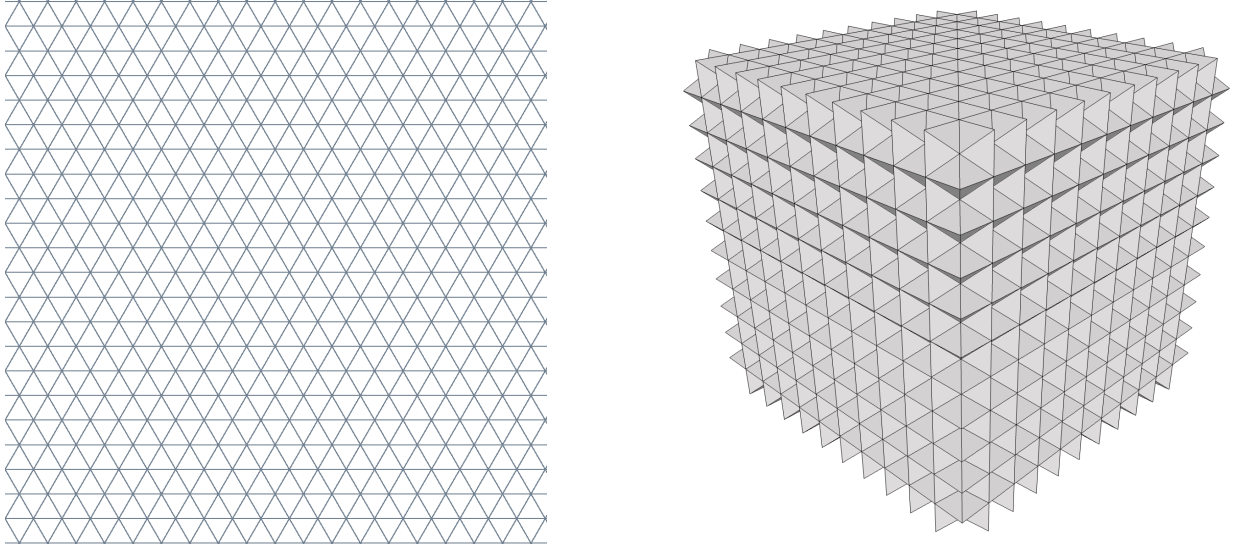


Figure F.4: **Grid structure to initialize real values in 2D (left) and 3D (right).** Every face in the grid structure satisfies *Minimum-Ball* condition (Definition 7.3.1).

F.2.2.2 Step 2: Position Optimization

In this step, we fix the point-wise real values (ψ) and optimize only the point positions. For clarity, we formally describe the process in [algorithm 6](#), and explain the algorithm line by line below:

- **Lines 1-2:** For the given set of points, we denote their positions as $\mathbb{P}$ and their real values as Ψ .
- **Line 3:** We define the total number of optimization steps as n_0 .
- **Line 4:** We define the number of optimization steps required to refresh query faces and their nearest neighbor cache as n_1 . Since point positions are optimized, the point configuration evolves during optimization, potentially leading to the emergence of new faces that were previously unobservable. To account for these changes, we refresh the query faces periodically.
- **Line 5:** We denote the number of nearest neighbors to store in the cache for the query faces as K .

- **Lines 6-7:** The optimization process runs for n_0 steps.
- **Lines 8-12:** At every n_1 step, we update the query faces based on the current point configuration.

In the `Update-Query-Faces` function, which uses point positions and their real values, we:

- Extract points with a real value of 1.
- For each extracted point, find its 10-nearest neighbors that also have a real value of 1, since any face containing a point with a real value of 0 is considered non-existent.
- Perform Delaunay Triangulation (DT) for the entire point set and collect faces in DT where all points have a real value of 1. This ensures the inclusion of as many faces as possible during optimization, helping to eliminate potential holes later.

For the updated query faces, we compute the centers of their minimum bounding balls. Subsequently, we identify the K -nearest neighbors of these centers in $\mathbb{P}$ and store this information in the nearest neighbor cache $\mathbb{C}$.

- **Lines 13-16:** Using the current point configuration, we compute the minimum bounding balls ($B_{\mathbb{F}}$) for the query faces. For each bounding ball center, we find the nearest neighbor in the nearest neighbor cache $\mathbb{C}$ by calculating the distances to points in $\mathbb{C}$ and selecting the closest one. We then compute the signed distance $d(B_{\mathbb{F}}, \mathbb{P})$ for the query faces and use it to get the probability $\lambda_{min}(\mathbb{F})$.
- **Line 17:** Since the query faces consist only of points with a real value of 1, we set the final face probability $\lambda(\mathbb{F})$ to be the same as $\lambda_{min}(\mathbb{F})$ ([section 7.3.1](#)).
- **Line 18:** Based on the point positions, query faces, and their existence probabilities, we compute the loss L to minimize following [eq. \(F.6\)](#).

Algorithm 6 Position Optimization

```
1:  $\mathbb{P}, \Psi \leftarrow$  Set of points and their real values
2:  $\alpha_{min} \leftarrow$  Coefficient for sigmoid function
3:  $n_0 \leftarrow$  Number of optimization steps
4:  $n_1 \leftarrow$  Number of refresh steps for query faces
5:  $K \leftarrow$  Number of nearest neighbors to store in cache
6:  $i \leftarrow 0$ 
7: while  $i < n_0$  do
8:   if  $i \bmod n_1 = 0$  then
9:      $\mathbb{F} \leftarrow$  Update-Query-Faces( $\mathbb{P}, \Psi$ )
10:     $B_{\mathbb{F}}^c, B_{\mathbb{F}}^r \leftarrow$  Compute-Minimum-Ball( $\mathbb{P}, \mathbb{F}$ )
11:     $\mathbb{C} \leftarrow$  Find-KNN( $B_{\mathbb{F}}^c, \mathbb{P}, K$ )
12:  end if
13:   $B_{\mathbb{F}}^c, B_{\mathbb{F}}^r \leftarrow$  Compute-Minimum-Ball( $\mathbb{P}, \mathbb{F}$ )
14:   $P_{\mathbb{F}}^{nearest} \leftarrow$  Find-NN-CACHE( $B_{\mathbb{F}}^c, \mathbb{C}$ )
15:   $d(B_{\mathbb{F}}, \mathbb{P}) \leftarrow B_{\mathbb{F}}^r - \|P_{\mathbb{F}}^{nearest} - B_{\mathbb{F}}^c\|$ 
16:   $\lambda_{min}(\mathbb{F}) \leftarrow \sigma(d(B_{\mathbb{F}}, \mathbb{P}) \cdot \alpha_{min})$ 
17:   $\lambda(\mathbb{F}) \leftarrow \lambda_{min}(\mathbb{F})$ 
18:   $L \leftarrow$  Compute-Loss( $\mathbb{P}, \mathbb{F}, \lambda(\mathbb{F})$ )
19:  Update  $\mathbb{P}$  to minimize  $L$ 
20:   $i \leftarrow i + 1$ 
21: end while
```

- **Lines 19-20:** Finally, we update the point positions $\mathbb{P}$ to minimize L and iterate the process.

F.2.2.3 Step 3: Real Value Optimization

In this step, we re-optimize the point-wise real values while keeping the point positions fixed. From the current point configuration, we identify all faces in the Delaunay Triangulation (DT) that satisfy the *Minimum-Ball* condition. Note that any face satisfying this condition must exist in the DT (Lemma 7.3.2). Thus, we first compute the DT of the points and then verify whether each face in the DT satisfies the *Minimum-Ball* condition.

Next, we follow a similar optimization process to Step 1 (Appendix F.2.2.1). Additionally, if it was the multi-view reconstruction task, we remove invisible faces to remove redundant faces as much as possible. If this was the last epoch, we return the post-processed mesh.

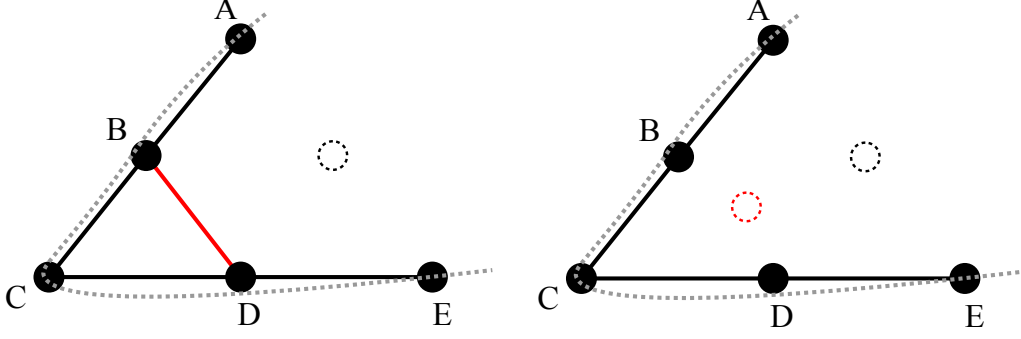
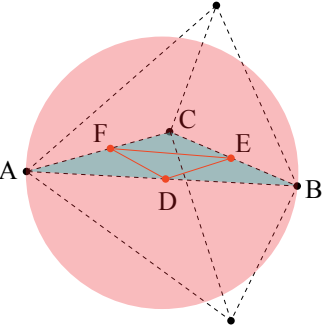


Figure F.5: **Point insertion for removing undesirable face.** (Left) To reconstruct the ground truth shape, we need to set the real value (ψ) of points A-E to 1. The point rendered with dotted line has real value of 0. Then, we observe unnecessary face $\overline{BD}$ exists. (Right) To remove this face, we insert **additional point** that carries 0 real value near the unnecessary face.

F.2.2.4 Step 4: Subdivision

To reconstruct fine geometric details of the given shape, we subdivide the current mesh mainly by adding points with $\psi = 1$ at the middle of edges that are adjacent to currently existing faces. In the inset, points A, E, D, F are the newly inserted points. They form 4 sub-faces with A, B, C , and they all satisfy Definition 7.3.1. Therefore, we can guarantee that these sub-faces will exist at the start of next epoch. Note that this guarantee does not hold for WDT.



At the same time, it is also possible to insert additional points into faces that *should not* exist in the next epoch, effectively removing such faces at the start of the next epoch. For example, during the real value optimization step in the pipeline (fig. 7.6, Appendix F.2.2.3), invisible faces are removed for multi-view reconstruction task. After optimization, we may observe removed faces with all their points have a real value of 1.0, creating a contradiction. This situation could arise due to ambiguities in the mesh definition, as illustrated in fig. F.5.

To eliminate these undesirable faces, additional points with a real value of 0 are inserted at their

circumcenters. Consequently, after subdivision, several holes may appear on the surface because these additional points might also be inserted into faces that *should* exist (fig. 7.6). However, most of these holes are resolved during subsequent optimization steps.

F.3 Experimental Details and Additional Results

In this section, we outline the experimental settings used for the results in section 7.5 and present additional results to support our claims.

F.3.1 Dataset

Here, we provide details on the datasets described in section 7.5.2.

F.3.1.1 Font

We used four font styles: Pacifico, Permanent-Marker, Playfair-Display, and Roboto.

F.3.1.2 Thingi10K

We manually selected 10 closed surfaces and 10 open surfaces from the Thingi10K dataset [1]. Specifically, we used the following models, denoted by their file IDs:

- **Closed surfaces:** 47926, 68380, 75147, 80650, 98576, 101582, 135730, 274379, 331105, and 372055.
- **Open surfaces:** 40009, 41909, 73058, 82541, 85538, 131487, 75846, 76278, 73421, and 106619.

These models were chosen because they exhibit minimal self-occlusions, enabling dense observations from multi-view images. Additionally, we randomly selected 500 models and used for comparisons.

F.3.1.3 Objaverse

We manually selected 30 mesh models that exhibit diverse topology from Objaverse [2], which include both closed and open surfaces, and also have small scenes. Some of these models are rendered in [figs. F.7, F.8, 7.3, 7.6 and 7.9](#).

F.3.2 2D Point Cloud Reconstruction

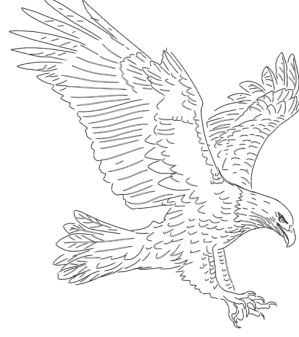
F.3.2.1 Hyperparameters

DMesh++

- Learning rate (real value, ψ): 0.3
- Learning rate (position): 0.001
- Number of epochs: 1
- Number of optimization steps
 - Step 1 (Real value initialization): 100
 - Step 2 (Point position optimization): 500



(a) Flower, # Edge = 99K, 6 min.



(b) Eagle, # Edge = 179K, 11 min.



(c) Picasso, # Edge = 159K, 8 min.



(d) Egyptian, # Edge = 227K, 19 min.



(e) Chinese, # Edge = 987K, 86 min.

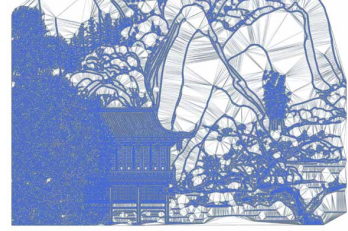


Figure F.6: 2D point cloud reconstruction result for complex drawings. For each drawing, we report both the number of edges and the reconstruction time. For the Chinese drawing, we additionally render the “imaginary” part on the right to clearly illustrate its complexity.

F.3.2.2 Reconstruction of Complex Drawings

In [fig. F.6](#), we present the reconstruction results for complex 2D drawings. As the figure illustrates, DMesh++ successfully reconstructs intricate 2D geometries from point clouds, even when the number of edges approaches nearly 1 million.

F.3.3 3D Point Cloud Reconstruction

F.3.3.1 Hyperparameters

DMesh++

- Initial Grid Edge Length: $3 \times$ input point cloud density

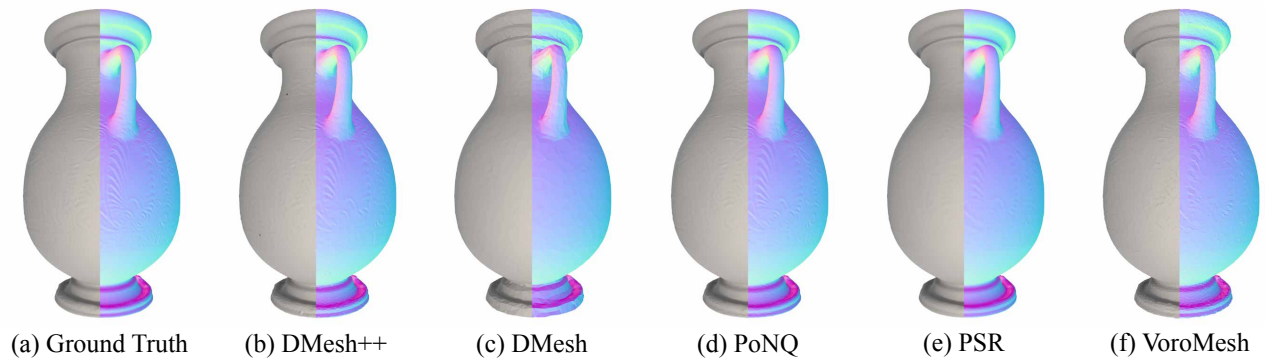


Figure F.7: **Qualitative comparison of 3D point cloud reconstruction results for a closed surface (vase).** For each image, we render the view-point normal on the right, and the diffuse image on the left. Among the baseline methods that reconstruct watertight mesh from point clouds, PoNQ [26] performs the best in reconstructing fine geometric details. While DMesh [20] fails at reconstructing such details due to the lack of mesh complexity, DMesh++ successfully recovers them and produce comparable result to PoNQ.

- Learning rate (position): 0.001
- Number of epochs: 1
- Number of optimization steps
 - Step 2 (Point position optimization): 2000
 - Step 3 (Real value optimization): 0

F.3.4 3D Multi-View Reconstruction

F.3.4.1 Hyperparameters

Remeshing [3]

- Image Batch Size: 8
- Number of Optimization Steps: 1000

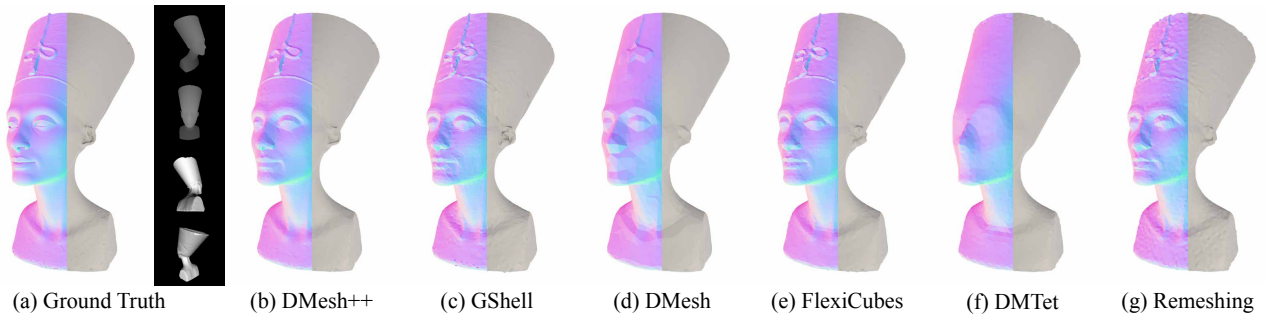


Figure F.8: **Qualitative comparison of 3D multi-view reconstruction results for a closed surface (sculpture).** We render the input diffuse and depth images alongside the ground truth image. For each image, we render the view-point normal on the left, and the diffuse image on the right. We can observe that the reconstruction result of DMesh++ is as good as the other baseline methods that are optimized for closed surfaces.

- Learning Rate: 0.1
- Edge Length Limits: [0.02, 0.15]

The "Edge Length Limits" were adjusted to produce meshes with a similar number of vertices and faces to other methods for a fair comparison.

DMTet [9]

- Image Batch Size: 8
- Number of Optimization Steps: 5000
- Learning Rate: 0.001
- Grid Resolution: 128

The SDF was initialized to a sphere, as in the original implementation, before starting optimization.

FlexiCubes [11]

- Image Batch Size: 8

- Number of Optimization Steps: 2000
- Number of Warm-up Steps: 1500
- Learning Rate: 0.01
- Grid Resolution: 80
- Triangle Aspect Ratio Loss Weight: 0.01

The SDF was initialized randomly, following the original implementation. To improve the quality of the output mesh, we adopted a triangle aspect ratio loss, designed to minimize the average aspect ratio of triangles in the mesh. The mesh was first optimized for 1500 steps as a warm-up without the triangle aspect ratio loss, followed by 500 steps with the additional loss.

Additionally, we observed that the output mesh often included false internal structures, which significantly degraded the Chamfer Distance (CD) compared to the ground truth mesh. To mitigate this, we performed a visibility test on the output mesh to remove these false internal structures as much as possible.

GShell [12]

- Image Batch Size: 8
- Number of Optimization Steps: 5000
- Number of Warm-up Steps: 4500
- Learning Rate: 0.01
- Grid Resolution: 80

- Triangle Aspect Ratio Loss Weight: 0.0001

To enhance the quality of the output mesh, we employed the same additional measures as FlexiCubes. We found that longer optimization steps were required for GShell compared to FlexiCubes to effectively handle open surfaces.

DMesh++ Settings

- Initial Grid Edge Length: 0.05
- Learning Rate (Real Value, ψ): 0.01
- Learning Rate (Position): 0.001
- Number of Epochs: 2
 - Image Res. / Batch Size at Epoch 1: (256, 256), 1
 - Image Res. / Batch Size at Epoch 2: (512, 512), 1
- Number of Optimization Steps
 - Step 1 (Real Value Initialization): 1000
 - Step 2 (Point Position Optimization): 2000
 - Step 3 (Real Value Optimization): 1000

In the first epoch, we used lower-resolution images as part of a coarse-to-fine approach.

F.3.4.2 Limitations

Despite DMesh++’s success in reconstructing geometrically accurate meshes from multi-view images (of a synthetic object or scene), it currently cannot recover meshes from real-world images.

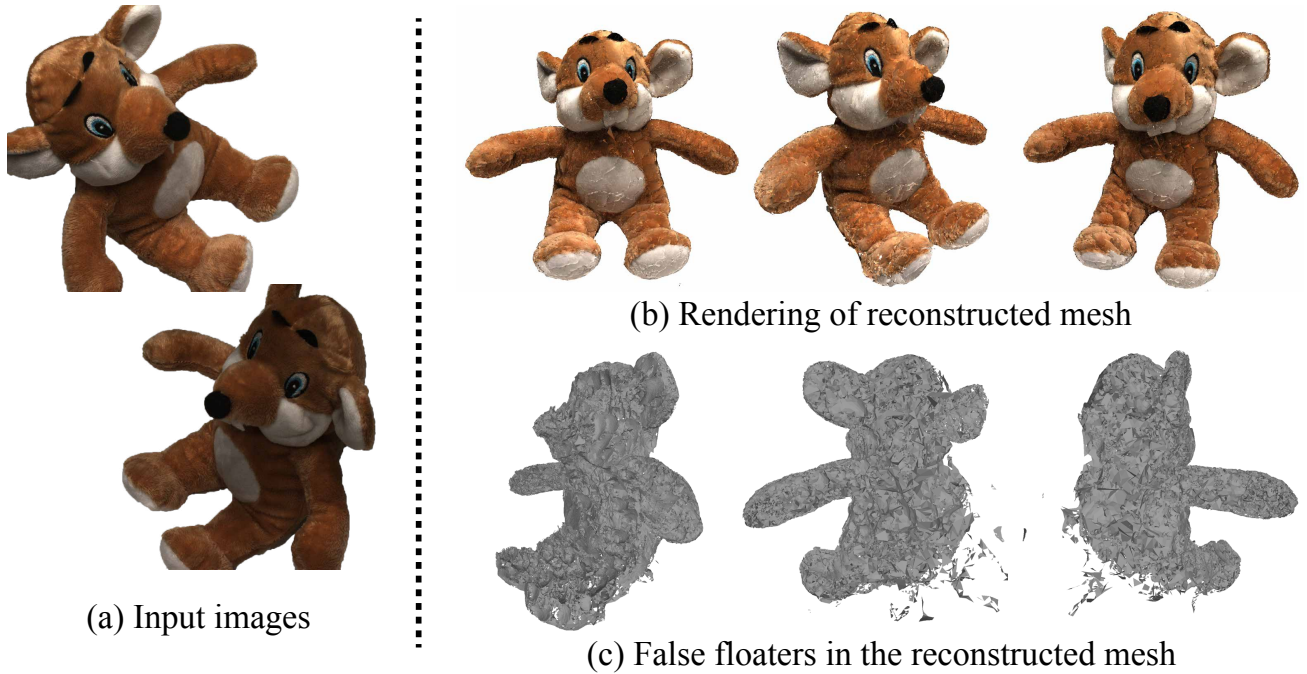


Figure F.9: **3D reconstruction from real-world images in DTU dataset [27]**. The input images are shown on left, and the reconstructed mesh is shown on right.

This limitation stems from an inadequate rendering model — our current per-vertex color model is too simple to capture detailed geometry, and our algorithm assumes full knowledge of lighting conditions, which is not available in real-world scenarios.

To illustrate this, we applied our reconstruction algorithm to real-world images from the DTU dataset [27], as shown in [fig. F.9](#). While our method approximates the real-world images ([fig. F.9\(b\)](#)), the extracted mesh exhibits numerous false floaters ([fig. F.9\(c\)](#)). We believe this suboptimal result is due to the lack of proper rendering models and regularizations, and addressing this issue by integrating DMesh++ with other reconstruction mechanisms is an exciting direction for future research, as discussed in [section 7.6](#).

F.4 *Reinforce-Ball* algorithm

Here we introduce an experimental algorithm that further enhances DMesh++’s capability. As discussed in [section 7.3.1](#), DMesh++ no longer uses the per-point weights found in DMesh [20]. In DMesh, optimizing per-point weights helps control mesh complexity: stronger regularization on these weights results in a simpler output mesh. Since DMesh++ lacks this mechanism, it cannot directly regulate mesh complexity during optimization. To address this limitation, we propose the *Reinforce-Ball* algorithm, which reduces unnecessary faces while preserving essential geometric details.

F.4.1 Local Minima of Weight Regularization

Before delving into the details of the Reinforce-Ball algorithm, we first highlight a limitation of DMesh’s per-point weight regularization. Specifically, while per-point weights are relevant for controlling mesh complexity, they alone cannot achieve adaptive resolution or produce a mesh that is both efficient and precise. Below, we explain the reasons in detail, assuming that per-point probabilities are optimized and that the *Minimum-Ball* condition is employed to compute face probabilities.

In [fig. F.10](#), we provide an example in a rendering scenario. A camera is placed on the left, and three different probabilistic meshes are shown on the right.

In **case (1)**, there are three points: A, B, and C. By connecting points A and B, the ground truth shape can be perfectly reconstructed, making point C redundant. Assume the optimization starts from this state, where all points have an existence probability of 1.0. According to the *Minimum-Ball* condition, the probabilities of faces $\overline{AC}$ and $\overline{BC}$ will also be 1.0. In this scenario, if a ray from the camera intersects the mesh, the accumulated opacity will be 1.0, representing a fully opaque surface. Consequently, the reconstruction loss will be 0.0, as the fully opaque faces perfectly match the ground

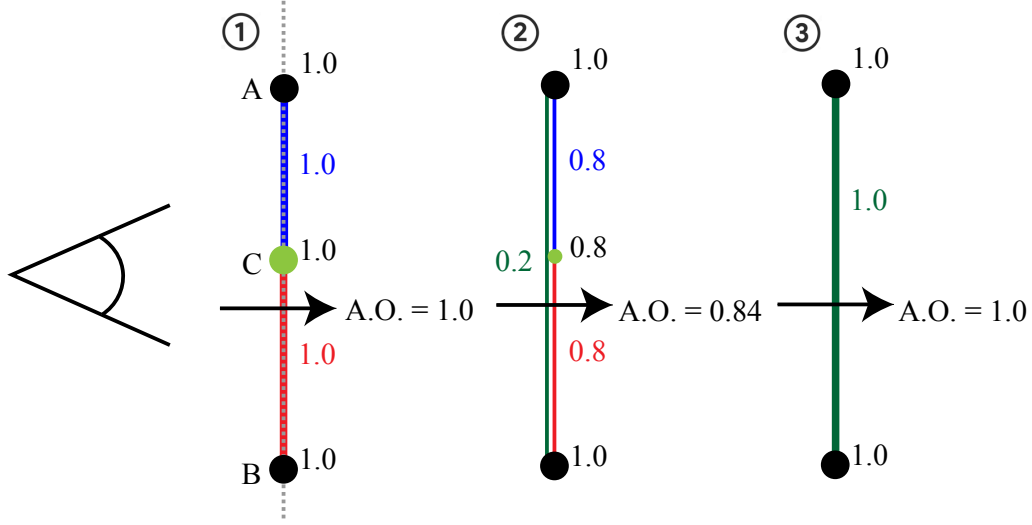


Figure F.10: **Local minima of weight regularization in a rendering setting.** (1) The ground truth geometry is rendered in gray dotted line. There are 3 points (A, B, C), where only the end points (A, B) are necessary for fully representing the underlying shape. Every point has weight 1.0, which is written to the next of each point. In this case, faces $\overline{AC}$ and $\overline{BC}$ exist with probability 1.0, which corresponds to their opacity. In this case, for a ray that goes through this mesh, the accumulated opacity (A.O.) becomes 1.0, and the reconstruction loss is 0. (2) When weight regularization reduces the weight of (redundant) C to 0.8, the probability of faces $\overline{AC}$ and $\overline{BC}$ becomes 0.8, and that of $\overline{AB}$ becomes 0.2. However, in this case, the accumulated opacity of the same ray becomes 0.84, which results in non-zero reconstruction loss. (3) Therefore, with a small weight regularization, we cannot remove C to get this optimal mesh, which contains only $\overline{AB}$, and attains 0 reconstruction loss.

truth.

In **case (3)**, the optimal configuration is rendered, where the redundant point C is removed. The probability of face $\overline{AB}$ becomes 1.0, making it fully opaque. Again, the reconstruction loss is 0.0.

In **case (2)**, an intermediate state between cases (1) and (3) is rendered. Assume that the probability of point C is reduced to 0.8 due to regularization. Consequently, the probabilities of faces $\overline{AC}$ and $\overline{BC}$ are also reduced to 0.8 because one of their endpoints, C, has a probability of 0.8. Simultaneously, the probability of face $\overline{AB}$ increases from 0 to 0.2, as the probability of point C, which lies inside the minimum bounding ball of the face, is 0.8.

Now, consider a camera ray passing through $\overline{AB}$ and $\overline{BC}$ sequentially (the order does not matter due to their tight overlap). Using alpha blending, the accumulated opacity is computed as:

$$\text{Accumulated Opacity: } 0.2 + (1.0 - 0.2) \cdot 0.8 = 0.84. \quad (\text{F.11})$$

This calculation shows that the accumulated opacity is reduced to 0.84.

The key issue arises from the **dependency** between the probabilities of $\overline{AB}$, $\overline{AC}$, and $\overline{BC}$. In the above formulation, the term $(1.0 - 0.2) \cdot 0.8$ represents the probability that the ray misses $\overline{AB}$ and hits $\overline{BC}$. If the probabilities of $\overline{AB}$ and $\overline{BC}$ were independent, this formulation would be correct. However, they are dependent: in fact, the probability of $\overline{BC}$ equals $1.0 - \overline{AB}$ because both depend on the probability of C. Thus, the actual accumulated opacity should be:

$$0.2 + (1.0 - 0.2) \cdot 1.0 = 1.0. \quad (\text{F.12})$$

However, the alpha blending technique used here does not account for such dependencies, leading to a reduction in accumulated opacity. This reduction artificially increases the reconstruction loss. To minimize the loss, the optimizer increases the probability of C again, preventing convergence to the optimal case (3).

This dependency issue creates a local minimum that the previous formulation cannot overcome. This is why we propose the *Reinforce-Ball* algorithm.

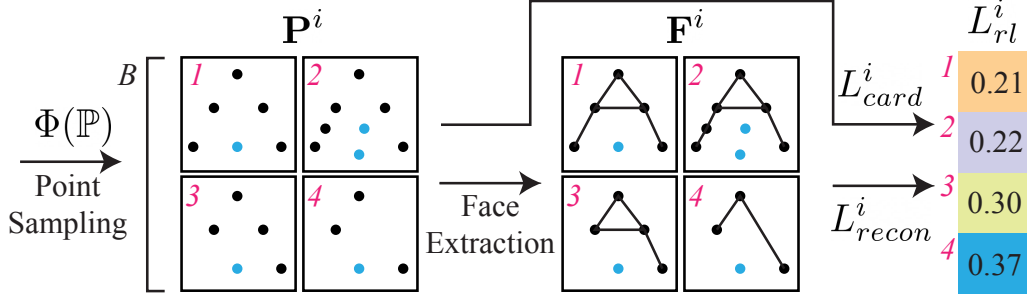


Figure F.11: **Overview of Reinforce-Ball Algorithm.** Based on per-point existence probability ($\Phi(\mathbb{P})$), we sample points for B number of batches ($\mathbf{P}^i$). Here we use $B = 4$, and assume we are reconstructing shape “A”. The points with $\psi = 1$ are rendered in black, while those with $\psi = 0$ are rendered in blue. Then, we identify existing faces in each batch ($\mathbf{F}^i$) based on [eq. \(7.2\)](#). With $\mathbf{P}^i$ and $\mathbf{F}^i$, we compute loss for each batch. Note that the case (1, 2) are better than (3, 4), because they reconstruct the shape better (L_{recon}^i). Also, the case (1) is better than (2), because it has less number of points (L_{card}^i). To minimize the expected loss ($\mathbb{E}[L_{rl}]$), we should maximize the probability to sample the case 1. We optimize $\Phi(\mathbb{P})$ to do that.

F.4.2 Algorithm Overview

In the *Reinforce-Ball* algorithm, we define per-point existence probability and optimize it using stochastic optimization technique [\[169\]](#). The overview of this algorithm is given in [fig. F.11](#).

To elaborate, for a point $p \in \mathbb{P}$, let us denote the probability of it as $\phi(p) \in [0, 1]$, and concatenation of them as $\Phi(\mathbb{P})$. Then, assuming we sample points independently, we can sample a set of points $\mathbf{P}$ from $\Phi(\mathbb{P})$ and compute its probability as follows:

$$P(\mathbf{P}|\Phi(\mathbb{P})) = \prod_{p \in \mathbf{P}} \phi(p) \cdot \prod_{p \in \mathbb{P} - \mathbf{P}} (1 - \phi(p)). \quad (\text{F.13})$$

Now, we sample points for B batches, and denote the sample points for i -th batch as $\mathbf{P}^i$. Based on $\mathbf{P}^i$ and tessellation function in [eq. \(7.2\)](#), we can find out which faces exist for the i -th batch. Importantly, this process does not require evaluating all possible global face combinations; instead, it

focuses only on local combinations, leveraging the *minimum-ball* condition in the tessellation function. We write these faces as $\mathbf{F}^i$, and use them for computing reconstruction loss for i -th batch (L_{recon}^i). We also compute “cardinality” loss for i -th batch (L_{card}^i), which is just the number of sampled points ($|\mathbf{P}^i|$). Then, we can compute the loss L_{rl}^i as

$$L_{rl}^i = L_{recon}^i + \epsilon_{card} \cdot L_{card}^i, \quad (\text{F.14})$$

where ϵ_{card} is a small tunable hyperparameter to adjust the weight of the cardinality loss. If we write the final loss for a set of sampled points $\mathbf{P}$ as $L_{rl}(\mathbf{P})$, we aim at minimizing the expected loss:

$$\mathbb{E}_{\mathbf{P} \sim \Phi(\mathbb{P})} L_{rl}(\mathbf{P}) = \sum P(\mathbf{P} | \Phi(\mathbb{P})) \cdot L_{rl}(\mathbf{P}). \quad (\text{F.15})$$

F.4.3 Formal Definition

In [algorithm 7](#), we formally describe the *Reinforce-Ball* algorithm in detail:

- **Line 1:** In the *Reinforce-Ball* algorithm, we optimize per-point probabilities for n_0 epochs, with each epoch consisting of n_1 optimization steps. In our experiments, we set $n_0 = 10$ and $n_1 = 2000$.
- **Line 2:** We define the number of batches used during optimization as B . Increasing B improves the stability of the gradient computation but also increases computational cost. In our experiments, we set $B = 1024$.
- **Line 3:** Initialize the per-point probability of every point to 0.99, as all points are assumed to exist with high probability before optimization. The probabilities are not set to 1.0 to avoid

Algorithm 7 Reinforce-Ball

```
1:  $n_0, n_1 \leftarrow$  Number of epochs and optimization steps
2:  $B \leftarrow$  Number of batch samples
3:  $\Phi \leftarrow$  Per-point probabilities, initialized to 0.99
4:  $i \leftarrow 0$ 
5: while  $i < n_0$  do
6:    $\mathbb{F} \leftarrow$  Update-Query-Faces( $\mathbb{P}, \Psi$ )
7:    $B_{\mathbb{F}} \leftarrow$  Compute-Minimum-Ball( $\mathbb{P}, \mathbb{F}$ )
8:    $j \leftarrow 0$ 
9:   while  $j < n_1$  do
10:    ( $k = 1, \dots, B$ )
11:     $\mathbf{P}^k \leftarrow$  Sample-Points( $\mathbb{P}, \Phi$ )
12:     $\mathbf{F}^k \leftarrow$  Get-Exist-Faces( $\mathbf{P}^k, \mathbb{F}, B_{\mathbb{F}}$ )
13:     $L_{rl}^k \leftarrow$  Compute-Loss( $\mathbb{P}, \mathbf{P}^k, \mathbf{F}^k$ )
14:     $\frac{\partial \mathbb{E}[L_{rl}]}{\partial \Phi} \leftarrow$  Estimate-Gradient( $\Phi, \mathbf{P}^k, L_{rl}^k$ )
15:     $\Phi \leftarrow$  Update-Gradient( $\Phi, \frac{\partial \mathbb{E}[L_{rl}]}{\partial \Phi}$ )
16:  end while
17:   $\mathbb{P}, \Psi \leftarrow$  Get-Remaining-Points( $\mathbb{P}, \Psi, \Phi$ )
18: end while
```

every sampled batch (Line 11) including all points, which would prevent optimization from progressing.

- **Lines 4-5:** Perform multiple epochs of optimization.
- **Line 6:** Gather the possibly existing faces ($\mathbb{F}$) based on the current point configuration and their real values. This function is the same as the one used in the Point Optimization step (Appendix [F.2.2.2](#)).
- **Line 7:** Compute the minimum bounding ball $B_{\mathbb{F}}$ for the gathered query faces.
- **Lines 8-9:** Perform the optimization steps within the current epoch.
- **Line 10:** Consider B batches, each containing a different point configuration based on the sampled points.

Method (hyperparameter)	CD($\times 10^{-6}$) $\downarrow$	# Verts.	# Edges.	Time (sec)
DMesh [20] (0)	1.97	2506	2245	30.39
DMesh (10^{-4})	2.68	666	693	153.10
DMesh (10^{-3})	12.48	456	488	152.37
DMesh++ (0)	1.82	2862	2793	11.33
DMesh++ (10^{-6})	1.86	1386	1394	278.88
DMesh++ (10^{-5})	2.77	149	152	200.05

Table F.1: **Quantitative ablation studies on Reinforce-Ball algorithm.** As we increase ϵ_{card} (in parenthesis) for DMesh++, we can significantly reduce the mesh complexity without losing geometric details, while DMesh cannot do the same with λ_{weight} .

- **Line 11:** For each batch, sample points from $\mathbb{P}$ based on their probabilities Φ . Each point is sampled independently, and the probability of sampling a specific batch is computed as shown in eq. (F.13). The sampled points in the k -th batch are denoted as $\mathbf{P}^k$.
- **Line 12:** For each batch, determine the existing faces in $\mathbb{F}$ based on the sampled points. Specifically, a face F exists if all its points are included in the sampled points and its B_F satisfies the *Minimum-Ball* condition. The existing faces in the k -th batch are denoted as $\mathbf{F}^k$.
- **Line 13:** For each batch, compute the loss as the sum of the reconstruction loss (L_{recon}) and the cardinality loss (L_{card}), as discussed in Appendix F.4.2.
- **Line 14:** Estimate the gradient of the expected loss ($\mathbb{E}[L_{rl}]$) with respect to the per-point probabilities Φ using the log-derivative trick [169]:

$$\nabla_{\Phi} \mathbb{E}_{\mathbf{P} \sim \Phi} [L_{rl}] \approx \frac{1}{B} \sum_{i=1}^B \nabla_{\Phi} \log P(\mathbf{P}^i | \Phi) \cdot L_{rl}^i. \quad (\text{F.16})$$

To reduce the variance of the gradient, we normalize L_{rl} across the batch before the computation [32].

- **Line 15:** Update Φ using the estimated gradients.

- **Line 17:** After completing an epoch, discard points whose probability is below a specified threshold. In our experiments, we set the threshold to 0.5. The remaining points are used for the next epoch. As points are removed, the query faces updated in Line 6 for the next epoch will span a larger area than in the previous epoch.

F.4.4 Experimental Results

Using the Reinforce-Ball algorithm, we can reconstruct efficient 2D meshes from point clouds that adapt to local geometry. As described in [section 7.5.2.1](#), we conducted 2D point cloud reconstruction experiments on the font dataset.

In [table F.1](#), we present quantitative ablation studies on the *Reinforce-Ball* algorithm. Increasing the tunable hyperparameter ϵ_{card} ([Appendix F.4.2](#)), which controls regularization strength, leads to a rapid reduction in vertices and edges. For instance, with $\epsilon_{card} = 10^{-5}$, edges decrease by nearly 94% with minimal impact on reconstruction quality. DMesh [20] also offers a tunable parameter, λ_{weight} , for weight regularization to reduce mesh complexity. However, while edge reduction occurs, DMesh’s reconstruction quality degrades more quickly. At $\epsilon_{card} = 10^{-5}$, our method achieves a similar CD loss to DMesh with $\lambda_{weight} = 10^{-4}$ but uses about 78% fewer edges. This advantage is also evident in [fig. F.12](#), where our *Reinforce-Ball* algorithm removes redundant edges effectively and adapts the mesh to local geometry. In contrast, DMesh’s edge removal disregards local geometry, resulting in loss of detail.

Likewise, we successfully highlighted the limitation of DMesh’s weight regularization and demonstrated that the *Reinforce-Ball* algorithm can eliminate redundant mesh faces within the DMesh++ framework without sacrificing geometric details. However, since the method is not yet easily extensi-

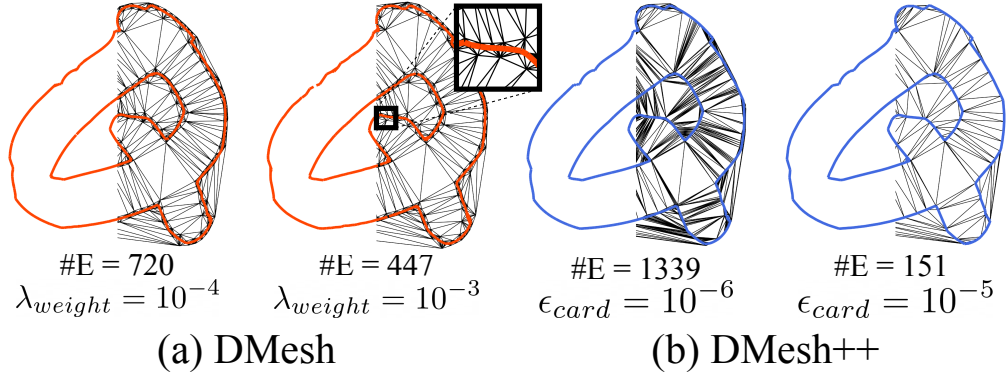


Figure F.12: **Qualitative ablation studies on Reinforce-Ball algorithm (for letter ‘Q’).** We render “imaginary” (black) part and “real part” (red, blue) together.

ble to 3D and incurs high computational costs, we include these results in the Appendix.

Appendix G: Differentiable Delaunay Tetrahedral Mesh Generation

G.1 Volumetric Marching Tetrahedra Algorithm

Algorithm 8 Volumetric MT

```
1:  $\mathbb{T}, S \leftarrow$  Tetrahedral grid and SDF
2:  $\mathbb{M} \leftarrow$  List of tetrahedra
3: while  $i < |\mathbb{T}|$  do
4:    $T^i \leftarrow i$ -th tetrahedron
5:    $p_1^i, p_2^i, p_3^i, p_4^i \leftarrow$  4 vertices of  $T^i$ 
6:    $s_1^i, s_2^i, s_3^i, s_4^i \leftarrow S(p_1^i), S(p_2^i), S(p_3^i), S(p_4^i)$ 
7:   if  $\min_j s_j^i > 0$  then
8:     Ignore
9:   end if
10:  if  $\max_j s_j^i < 0$  then
11:    Add  $T^i$  to  $\mathbb{M}$ 
12:  end if
13:  if  $(\max_j s_j^i) \cdot (\min_j s_j^i) < 0$  then
14:     $V^i \leftarrow \text{Inner-Approximate}(T^i, S)$ 
15:     $C^i \leftarrow \text{Compute-Centroid}(V^i)$ 
16:     $\mathbb{F}_{iso}^i \leftarrow \text{Original-MT}(T^i, S)$ 
17:    for Face  $F_j^i$  on  $T^i$  ( $j = 1, 2, 3, 4$ ) do
18:       $\mathbb{F}_j^i \leftarrow \text{Triangulate}(F_j^i, S)$ 
19:    end for
20:     $\mathbb{F}^i \leftarrow \mathbb{F}_{iso}^i \cup \mathbb{F}_1^i \cup \mathbb{F}_2^i \cup \mathbb{F}_3^i \cup \mathbb{F}_4^i$ 
21:    for  $F \in \mathbb{F}^i$  do
22:       $T^F \leftarrow \text{Merge } F \text{ and } C$ 
23:      Add  $T^F$  to  $\mathbb{M}$ 
24:    end for
25:  end if
26: end while
```

Bibliography

- [1] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016.
- [2] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023.
- [3] Werner Palfinger. Continuous remeshing for inverse rendering. *Computer Animation and Virtual Worlds*, 33(5):e2101, 2022.
- [4] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021.
- [5] 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. *arXiv preprint arXiv:2106.10689*, 2021.
- [6] Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.
- [7] Xiaoxiao Long, Cheng Lin, Lingjie Liu, Yuan Liu, Peng Wang, Christian Theobalt, Taku Komura, and Wenping Wang. Neuraludf: Learning unsigned distance fields for multi-view reconstruction of surfaces with arbitrary topologies. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20834–20843, 2023.
- [8] Yu-Tao Liu, Li Wang, Jie Yang, Weikai Chen, Xiaoxu Meng, Bo Yang, and Lin Gao. Neudf: Learning neural unsigned distance fields with volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 237–247, 2023.
- [9] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems*, 34:6087–6101, 2021.
- [10] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting triangular 3d models, materials, and lighting from images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8280–8290, 2022.

- [11] Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Transactions on Graphics (TOG)*, 42(4):1–16, 2023.
- [12] Zhen Liu, Yao Feng, Yuliang Xiu, Weiyang Liu, Liam Paull, Michael J Black, and Bernhard Schölkopf. Ghost on the shell: An expressive representation of general 3d shapes. *arXiv preprint arXiv:2310.15168*, 2023.
- [13] Marie-Julie Rakotosaona, Noam Aigerman, Niloy J Mitra, Maks Ovsjanikov, and Paul Guerrero. Differentiable surface triangulation. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021.
- [14] Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. *arXiv preprint arXiv:2204.07137*, 2022.
- [15] 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.
- [16] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.
- [17] Nissim Maruani, Roman Klokov, Maks Ovsjanikov, Pierre Alliez, and Mathieu Desbrun. Voromesh: Learning watertight surface meshes with voronoi diagrams. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14565–14574, 2023.
- [18] Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. A level set theory for neural implicit evolution under explicit flows. In *European Conference on Computer Vision*, pages 711–729. Springer, 2022.
- [19] Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. Neural dual contouring. *ACM Transactions on Graphics (TOG)*, 41(4):1–13, 2022.
- [20] Sanghyun Son, Matheus Gadelha, Yang Zhou, Zexiang Xu, Ming C Lin, and Yi Zhou. Dmesh: A differentiable mesh representation. *arXiv preprint arXiv:2404.13445*, 2024.
- [21] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.
- [22] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*, pages 347–353. 1998.
- [23] Akio Doi and Akio Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems*, 74(1):214–224, 1991.
- [24] 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):1–14, 2020.

- [25] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, Guido Ranzuglia, et al. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136. Salerno, Italy, 2008.
- [26] Nissim Maruani, Maks Ovsjanikov, Pierre Alliez, and Mathieu Desbrun. Ponq: a neural gem-based mesh representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3647–3657, 2024.
- [27] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 406–413, 2014.
- [28] Richard S Sutton, Andrew G Barto, et al. Introduction to reinforcement learning. 1998.
- [29] Ronald J Williams and Jing Peng. Reinforcement learning algorithms as function optimizers. In *Proceedings of the International Joint Conference on Neural Networks, Washington DC*, volume 2, pages 89–95, 1989.
- [30] Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.
- [31] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.
- [32] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.
- [33] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.
- [34] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.
- [35] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.
- [36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [37] Nicolas Heess, Dhruva Tb, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.
- [38] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

- [39] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.
- [40] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.
- [41] 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.
- [42] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.
- [43] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.
- [44] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- [45] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
- [46] M Mozer. A focused backpropagation algorithm. *Complex systems*, 3:349–381, 1989.
- [47] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [48] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.
- [49] 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. *arXiv preprint arXiv:2106.13281*, 2021.
- [50] Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. Pippis: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074. PMLR, 2018.
- [51] Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

- [52] Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.
- [53] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C Lin. Efficient differentiable simulation of articulated bodies. In *International Conference on Machine Learning*, pages 8661–8671. PMLR, 2021.
- [54] Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, pages 7805–7817. PMLR, 2021.
- [55] Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, page 6, 2019.
- [56] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. Add: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.
- [57] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. *Advances in neural information processing systems*, 31, 2018.
- [58] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.
- [59] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, pages 6265–6271. IEEE, 2019.
- [60] Marc A Toussaint, Kelsey Rebecca Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. 2018.
- [61] Mengxi Li, Rika Antonova, Dorsa Sadigh, and Jeannette Bohg. Learning tool morphology for contact-rich manipulation tasks with differentiable simulation. *arXiv preprint arXiv:2211.02201*, 2022.
- [62] Krishna Murthy Jatavallabhula, 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. *arXiv preprint arXiv:2104.02646*, 2021.
- [63] Rika Antonova, Jingyun Yang, Krishna Murthy Jatavallabhula, and Jeannette Bohg. Rethinking optimization with differentiable simulation from a global perspective. In *Conference on robot learning*, pages 276–286. PMLR, 2023.

- [64] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.
- [65] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.
- [66] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.
- [67] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.
- [68] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.
- [69] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.
- [70] Nicklas A Hansen, Hao Su, and Xiaolong Wang. Temporal difference learning for model predictive control. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162, pages 8387–8406. PMLR, 17–23 Jul 2022.
- [71] Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.
- [72] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on Robot Learning (CoRL)*, pages 2226–2240. PMLR, 2023.
- [73] Wei Zhu and Mitsuhiro Hayashibe. Autonomous navigation system in pedestrian scenarios using a dreamer-based motion planner. *IEEE Robotics and Automation Letters*, 8(6):3836–3843, 2023.
- [74] Younggyo Seo, Danijar Hafner, Hao Liu, Fangchen Liu, Stephen James, Kimin Lee, and Pieter Abbeel. Masked world models for visual control. In *Conference on Robot Learning*, pages 1332–1344. PMLR, 2023.
- [75] Patrick Lancaster, Nicklas Hansen, Aravind Rajeswaran, and Vikash Kumar. Modem-v2: Visuo-motor world models for real-world robot manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7530–7537. IEEE, 2024.
- [76] Yunhai Feng, Nicklas Hansen, Ziyang Xiong, Chandramouli Rajagopalan, and Xiaolong Wang. Finetuning offline world models in the real world. *arXiv preprint arXiv:2310.16029*, 2023.
- [77] Nicklas Hansen, Yixin Lin, Hao Su, Xiaolong Wang, Vikash Kumar, and Aravind Rajeswaran. Modem: Accelerating visual model-based reinforcement learning with demonstrations. *arXiv preprint arXiv:2212.05698*, 2022.

- [78] Martin Philip Bendsøe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer methods in applied mechanics and engineering*, 71(2):197–224, 1988.
- [79] Ole Sigmund and Kurt Maute. Topology optimization approaches: A comparative review. *Structural and multidisciplinary optimization*, 48(6):1031–1055, 2013.
- [80] Olivier Pironneau. On optimum profiles in stokes flow. *Journal of Fluid Mechanics*, 59(1):117–128, 1973.
- [81] Antony Jameson. Aerodynamic design via control theory. *Journal of scientific computing*, 3(3):233–260, 1988.
- [82] François-Xavier Le Dimet and Olivier Talagrand. Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects. *Tellus A: Dynamic Meteorology and Oceanography*, 38(2):97–110, 1986.
- [83] PHILIPPE Courtier, J-N Thépaut, and Anthony Hollingsworth. A strategy for operational implementation of 4d-var, using an incremental approach. *Quarterly Journal of the Royal Meteorological Society*, 120(519):1367–1387, 1994.
- [84] Rhys Newbury, Jack Collins, Kerry He, Jiahe Pan, Ingmar Posner, David Howard, and Akansel Cosgun. A review of differentiable simulators. *IEEE Access*, 2024.
- [85] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. *Advances in neural information processing systems*, 32, 2019.
- [86] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics. *ACM Transactions on Graphics (ToG)*, 41(2):1–21, 2021.
- [87] Yiling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Differentiable simulation of soft multi-body systems. *Advances in Neural Information Processing Systems*, 34:17123–17135, 2021.
- [88] Zhou Xian, Bo Zhu, Zhenjia Xu, Hsiao-Yu Tung, Antonio Torralba, Katerina Fragkiadaki, and Chuang Gan. Fluidlab: A differentiable environment for benchmarking complex fluid manipulation. *arXiv preprint arXiv:2303.02346*, 2023.
- [89] Tetsuya Takahashi, Junbang Liang, Yi-Ling Qiao, and Ming C Lin. Differentiable fluids with solid coupling for learning and control. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6138–6146, 2021.
- [90] 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. *Advances in neural information processing systems*, 32, 2019.

- [91] Samuel Schoenholz and Ekin Dogus Cubuk. Jax md: a framework for differentiable physics. *Advances in Neural Information Processing Systems*, 33:11428–11441, 2020.
- [92] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.
- [93] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.
- [94] Genesis Authors. Genesis: A generative and universal physics engine for robotics and beyond, December 2024.
- [95] Michael James Lighthill and Gerald Beresford Whitham. On kinematic waves ii. a theory of traffic flow on long crowded roads. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 229(1178):317–345, 1955.
- [96] Paul I Richards. Shock waves on the highway. *Operations research*, 4(1):42–51, 1956.
- [97] Harold J Payne. Model of freeway traffic and control. *Mathematical Model of Public System*, pages 51–61, 1971.
- [98] Gerald Beresford Whitham. *Linear and nonlinear waves*, volume 42. John Wiley & Sons, 2011.
- [99] AATM Aw and Michel Rascle. Resurrection of " second order" models of traffic flow. *SIAM journal on applied mathematics*, 60(3):916–938, 2000.
- [100] H Michael Zhang. A non-equilibrium traffic model devoid of gas-like behavior. *Transportation Research Part B: Methodological*, 36(3):275–290, 2002.
- [101] Jason Sewall, David Wilkie, Paul Merrell, and Ming C Lin. Continuum traffic simulation. In *Computer Graphics Forum*, volume 29, pages 439–448. Wiley Online Library, 2010.
- [102] Huan Yu and Miroslav Krstic. Traffic congestion control for aw–rascle–zhang model. *Automatica*, 100:38–51, 2019.
- [103] Arne Kesting, Martin Treiber, and Dirk Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.
- [104] Denos C Gazis, Robert Herman, and Renfrey B Potts. Car-following theory of steady-state traffic flow. *Operations research*, 7(4):499–505, 1959.
- [105] Denos C Gazis, Robert Herman, and Richard W Rothery. Nonlinear follow-the-leader models of traffic flow. *Operations research*, 9(4):545–567, 1961.
- [106] Gordon Frank Newell. Nonlinear effects in the dynamics of car following. *Operations research*, 9(2):209–229, 1961.

- [107] Peter G Gipps. A behavioural car-following model for computer simulation. *Transportation Research Part B: Methodological*, 15(2):105–111, 1981.
- [108] Masako Bando, Katsuya Hasebe, Akihiro Nakayama, Akihiro Shibata, and Yuki Sugiyama. Dynamical model of traffic congestion and numerical simulation. *Physical review E*, 51(2):1035, 1995.
- [109] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.
- [110] Rui Jiang, Qingsong Wu, and Zuojin Zhu. Full velocity difference model for a car-following theory. *Physical Review E*, 64(1):017101, 2001.
- [111] Laurent Magne, Sylvestre Rabut, and Jean-François Gabard. Towards an hybrid macro-micro traffic flow simulation model. In *INFORMS spring 2000 meeting*, 2000.
- [112] Emmanuel Bourrel and Jean-Baptiste Lesort. Mixing microscopic and macroscopic representations of traffic flow: Hybrid model based on lighthill–whitham–richards theory. *Transportation Research Record*, 1852(1):193–200, 2003.
- [113] Salim Mammar, Saïd Mammar, and Jean-Patrick Lebacque. Highway traffic hybrid macro-micro simulation model. *IFAC Proceedings Volumes*, 39(12):627–632, 2006.
- [114] 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.
- [115] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of sumo-simulation of urban mobility. *International journal on advances in systems and measurements*, 5(3&4), 2012.
- [116] Edouard Leurent. An environment for autonomous driving decision-making. <https://github.com/eleurent/highway-env>, 2018.
- [117] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.
- [118] Cathy Wu, Aboudy Kreidieh, Kanaad Parvate, Eugene Vinitzky, and Alexandre M Bayen. Flow: Architecture and benchmarking for reinforcement learning in traffic control. *arXiv preprint arXiv:1710.05465*, 10, 2017.
- [119] MATSim Association. The multi-agent transport simulation matsim.
- [120] PTV Group. Vissim: Multimodal traffic simulation software.
- [121] Philipp Andelfinger. Differentiable agent-based simulation for gradient-guided simulation-based optimization. In *Proceedings of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 27–38, 2021.
- [122] Jun Zhang, Wenxuan Ao, Junbo Yan, Can Rong, Depeng Jin, Wei Wu, and Yong Li. Moss: A large-scale open microscopic traffic simulation system. *arXiv preprint arXiv:2405.12520*, 2024.

- [123] Martin Treiber and Arne Kesting. Modeling lane-changing decisions with mobil. In *Traffic and Granular Flow '07*, page 211–221. Springer Berlin Heidelberg, 2009.
- [124] Cole Gulino, Justin Fu, Wenjie Luo, George Tucker, Eli Bronstein, Yiren Lu, Jean Harb, Xinlei Pan, Yan Wang, Xiangyu Chen, et al. Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research. *Advances in Neural Information Processing Systems*, 36, 2024.
- [125] Saman Kazemkhani, Aarav Pandya, Daphne Cornelisse, Brennan Shacklett, and Eugene Vinit-sky. Gpudrive: Data-driven, multi-agent driving simulation at 1 million fps. August 2024.
- [126] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Prad-han, Yuning Chai, Ben Sapp, Charles R. Qi, Yin Zhou, Zoey Yang, Aur’elien Chouard, Pei Sun, Jiquan Ngiam, Vijay Vasudevan, Alexander McCauley, Jonathon Shlens, and Dragomir Anguelov. Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9710–9719, October 2021.
- [127] 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. *Communi-cations of the ACM*, 65(1):99–106, 2021.
- [128] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and im-proving neural radiance fields. *arXiv:2010.07492*, 2020.
- [129] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020.
- [130] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022.
- [131] Anpei Chen, Zexiang Xu, Xinyue Wei, Siyu Tang, Hao Su, and Andreas Geiger. Dictionary fields: Learning a neural basis decomposition. *ACM Trans. Graph.*, 2023.
- [132] 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, 2020.
- [133] 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.
- [134] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit sur-faces and radiance fields for multi-view reconstruction. In *International Conference on Com-puter Vision (ICCV)*, 2021.
- [135] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deep-sdf: Learning continuous signed distance functions for shape representation. In *Proceed-ings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

- [136] Xinyue Wei, Fanbo Xiang, Sai Bi, Anpei Chen, Kalyan Sunkavalli, Zexiang Xu, and Hao Su. Neumanifold: Neural watertight manifold reconstruction with efficient and high-quality rendering support. *arXiv preprint arXiv:2305.17134*, 2023.
- [137] Zhengming Yu, Zhiyang Dou, Xiaoxiao Long, Cheng Lin, Zekun Li, Yuan Liu, Norman Müller, Taku Komura, Marc Habermann, Christian Theobalt, et al. Surf-d: High-quality surface generation for arbitrary topologies using diffusion models. *arXiv preprint arXiv:2311.17050*, 2023.
- [138] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346, 2002.
- [139] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *Advances in neural information processing systems*, 33:20154–20166, 2020.
- [140] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5799–5809, 2021.
- [141] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16123–16133, 2022.
- [142] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.
- [143] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 300–309, 2023.
- [144] Rui Chen, Yongwei Chen, Ningxin Jiao, and Kui Jia. Fantasia3d: Disentangling geometry and appearance for high-quality text-to-3d content creation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 22246–22256, 2023.
- [145] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation. *Advances in neural information processing systems*, 36:8406–8441, 2023.
- [146] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *ACM Siggraph Computer Graphics*, 24(5):63–70, 1990.
- [147] Kwan-Liu Ma and Thomas W Crockett. A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data. In *Proceedings of the IEEE symposium on Parallel rendering*, pages 95–ff, 1997.

- [148] Michael P Garrity. Raytracing irregular volume data. In *Proceedings of the 1990 workshop on Volume visualization*, pages 35–40, 1990.
- [149] Anis Benyoub and Jonathan Dupuy. Adaptive tetrahedral grids for volumetric path-tracing. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks*, pages 1–2, 2025.
- [150] Ken Museth and Santiago Lombeyda. Tetsplat: Real-time rendering and volume clipping of large unstructured tetrahedral meshes. In *IEEE Visualization 2004*, pages 433–440. IEEE, 2004.
- [151] Ryoichi Ando, Nils Thürey, and Chris Wojtan. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013.
- [152] Nuttapong Chentanez, Bryan E Feldman, François Labelle, James F O’Brien, and Jonathan R Shewchuk. Liquid simulation on lattice-based tetrahedral meshes. 2007.
- [153] Bryan M Klingner, Bryan E Feldman, Nuttapong Chentanez, and James F O’Brien. Fluid animation with dynamic meshes. In *ACM SIGGRAPH 2006 Papers*, pages 820–825. 2006.
- [154] Geoffrey Irving, Joseph Teran, and Ronald Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 131–140, 2004.
- [155] Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: a practitioner’s guide to theory, discretization and model reduction. In *Acm siggraph 2012 courses*, pages 1–50. 2012.
- [156] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 787–797. 2023.
- [157] Joseph Teran, Silvia S Blemker, Victor Ng-Thow-Hing, and Ronald Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Symposium on Computer Animation*, pages 68–74, 2003.
- [158] James F O’Brien and Jessica K Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 137–146, 1999.
- [159] Yi Zhou, Chenglei Wu, Zimo Li, Chen Cao, Yuting Ye, Jason Saragih, Hao Li, and Yaser Sheikh. Fully convolutional mesh autoencoder using efficient spatially varying kernels. *Advances in neural information processing systems*, 33:9251–9262, 2020.
- [160] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Advances in neural information processing systems*, 32, 2019.
- [161] 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.

- [162] Jun Gao, Wenzheng Chen, Tommy Xiang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. *Advances in neural information processing systems*, 33:9936–9947, 2020.
- [163] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, light, and material decomposition from images using monte carlo rendering and denoising. *Advances in Neural Information Processing Systems*, 35:22856–22869, 2022.
- [164] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. *Advances in neural information processing systems*, 35:31841–31854, 2022.
- [165] Ruowen Zhao, Zhengyi Wang, Yikai Wang, Zihan Zhou, and Jun Zhu. Flexidreamer: single image-to-3d generation with flexicubes. *arXiv preprint arXiv:2404.00987*, 2024.
- [166] Yiwei Ma, Yijun Fan, Jiayi Ji, Haowei Wang, Xiaoshuai Sun, Guannan Jiang, Annan Shu, and Rongrong Ji. X-dreamer: Creating high-quality 3d content by bridging the domain gap between text-to-2d and text-to-3d generation. *arXiv preprint arXiv:2312.00085*, 2023.
- [167] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
- [168] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.
- [169] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.
- [170] C Fu, C Fu, and Michael Michael. *Handbook of simulation optimization*. Springer, 2015.
- [171] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.
- [172] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 101:48, 2005.
- [173] Eugene Vinitsky, Aboudy Kreidieh, Luc Le Flem, Nishant Kheterpal, Kathy Jang, Cathy Wu, Fangyu Wu, Richard Liaw, Eric Liang, and Alexandre M Bayen. Benchmarks for reinforcement learning in mixed-autonomy traffic. In *Conference on robot learning*, pages 399–409. PMLR, 2018.
- [174] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

- [175] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.
- [176] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
- [177] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grand-master level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
- [178] Jing Yu Koh, Honglak Lee, Yinfei Yang, Jason Baldridge, and Peter Anderson. Pathdreamer: A world model for indoor navigation, 2021.
- [179] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.
- [180] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.
- [181] Yanchen Guan, Haicheng Liao, Zhenning Li, Jia Hu, Runze Yuan, Yunjian Li, Guohui Zhang, and Chengzhong Xu. World models for autonomous driving: An initial survey. *IEEE Transactions on Intelligent Vehicles*, 2024.
- [182] Niranjani Prasad, Li-Fang Cheng, Corey Chivers, Michael Draugelis, and Barbara E Engelhardt. A reinforcement learning approach to weaning of mechanical ventilation in intensive care units. *arXiv preprint arXiv:1704.06300*, 2017.
- [183] Anh N Nhu, Ngoc-Anh Le, Shihang Li, and Thang DV Truong. Physics-guided reinforcement learning system for realistic vehicle active suspension control. In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pages 422–429. IEEE, 2023.
- [184] Lei Yang, Zoltan Nagy, Philippe Goffin, and Arno Schlueter. Reinforcement learning for optimal control of low exergy buildings. *Applied Energy*, 156:577–586, 2015.
- [185] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870. PMLR, 2018.
- [186] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

- [187] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.
- [188] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.
- [189] Pierre Thodoroff, Wenyu Li, and Neil D. Lawrence. Benchmarking real-time reinforcement learning. In Samuel Albanie, João F. Henriques, Luca Bertinetto, Alex Hernández-García, Hazel Doughty, and Gül Varol, editors, *NeurIPS 2021 Workshop on Pre-registration in Machine Learning*, volume 181, pages 26–41. PMLR, 13 Dec 2022.
- [190] Nathan Lambert, Kristofer Pister, and Roberto Calandra. Investigating compounding prediction errors in learned dynamics models, 2022.
- [191] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning, 2019.
- [192] Vaisakh Shaj Kumar, Saleh Gholam Zadeh, Ozan Demir, Luiz Douat, and Gerhard Neumann. Multi time scale world models. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:26764–26775, 2023.
- [193] Michael Lutter, Leonard Hasenclever, Arunkumar Byravan, Gabriel Dulac-Arnold, Piotr Trochim, Nicolas Heess, Josh Merel, and Yuval Tassa. Learning dynamics models for model predictive agents, 2021.
- [194] John Charles Butcher. *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience, 1987.
- [195] C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, jan 1949.
- [196] Abdul J Jerri. The shannon sampling theorem—its various extensions and applications: A tutorial review. *Proceedings of the IEEE*, 65(11):1565–1596, 1977.
- [197] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning (CoRL)*, volume 100 of *Proceedings of Machine Learning Research*, pages 1094–1100. PMLR, 30 Oct–01 Nov 2020.
- [198] Xiangyuan Zhang, Weichao Mao, Saviz Mowlavi, Mouhacine Benosman, and Tamer Başar. Controlgym: Large-scale control environments for benchmarking reinforcement learning algorithms. In *6th Annual Learning for Dynamics & Control Conference*, pages 181–196. PMLR, 2024.

- [199] Sanghyun Son, Laura Zheng, Ryan Sullivan, Yi-Ling Qiao, and Ming Lin. Gradient informed proximal policy optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2024.
- [200] E Weinan. *Principles of multiscale modeling*. Cambridge University Press, 2011.
- [201] Zhexuan Zeng, Zuogong Yue, Alexandre Mauroy, Jorge Gonçalves, and Ye Yuan. A sampling theorem for exact identification of continuous-time nonlinear dynamical systems. *IEEE Transactions on Automatic Control*, pages 1–16, 2024.
- [202] Edward Lieberman and Ajay K Rathi. Traffic simulation. *Traffic flow theory*, 1997.
- [203] 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.
- [204] Michael J Cassidy and John R Windover. methodology for assessing dynamics of freeway traffic flow. 1995.
- [205] Carlos F Daganzo. Requiem for second-order fluid approximations of traffic flow. *Transportation Research Part B: Methodological*, 29(4):277–286, 1995.
- [206] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582. IEEE, 2018.
- [207] 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.
- [208] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph.*, 37(6), 2018.
- [209] Tao Du, Kui Wu, Andrew Spielberg, Wojciech Matusik, Bo Zhu, and Eftychios Sifakis. Functional optimization of fluidic devices with differentiable stokes flow. *ACM Trans. Graph.*, 2020.
- [210] Dan Cascaval, Mira Shalah, Phillip Quinn, Rastislav Bodik, Maneesh Agrawala, and Adriana Schulz. Differentiable 3d cad programs for bidirectional editing. *arXiv preprint arXiv:2110.01182*, 2021.
- [211] 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.
- [212] Siyuan Shen, Yang Yin, Tianjia Shao, He Wang, Chenfanfu Jiang, Lei Lan, and Kun Zhou. High-order differentiable autoencoder for nonlinear model reduction. *arXiv preprint arXiv:2102.11026*, 2021.

- [213] 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):1–14, 2021.
- [214] 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.
- [215] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *ICML*, 2020.
- [216] Yifei Li, Tao Du, Kui Wu, Jie Xu, and Wojciech Matusik. Diffcloth: Differentiable cloth simulation with dry frictional contact. *ACM Trans. Graph.*, 2022.
- [217] Philipp Holl, Vladlen Koltun, Kiwon Um, and Nils Thuerey. phiflow: A differentiable pde solving framework for deep learning via physical simulations. In *NeurIPS Workshop*, 2020.
- [218] Feixiang He, Yuanhang Xiang, Xi Zhao, and He Wang. Informative scene decomposition for crowd analysis, comparison and simulation guidance. *ACM Transactions on Graphics (TOG)*, 39(4):50–1, 2020.
- [219] Adèle Colas, Wouter van Toll, Katja Zibrek, Ludovic Hoyet, Anne-Hélène Olivier, and Julien Pettré. Interaction fields: Intuitive sketch-based steering behaviors for crowd simulation. In *Computer Graphics Forum*, 2022.
- [220] Torsten Hädrich, Daniel T Banuti, Wojtek Pałubicki, Sören Pirk, and Dominik L Michels. Fire in paradise: mesoscale simulation of wildfires. *ACM Transactions on Graphics (TOG)*, 40(4):1–15, 2021.
- [221] Yuko Ishiwaka, Xiao S Zeng, Michael Lee Eastman, Sho Kakazu, Sarah Gross, Ryosuke Mizutani, and Masaki Nakada. Foids: bio-inspired fish simulation for generating synthetic datasets. *ACM Transactions on Graphics (TOG)*, 40(6):1–15, 2021.
- [222] Randall J LeVeque et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.
- [223] Jason Douglas Sewall. *Efficient, scalable traffic and compressible fluid simulations using hyperbolic models*. PhD thesis, The University of North Carolina at Chapel Hill, 2011.
- [224] Nikolaus Hansen. The cma evolution strategy: a comparing review. *Towards a new evolutionary computation*, pages 75–102, 2006.
- [225] Dieter Kraft et al. A software package for sequential quadratic programming. 1988.
- [226] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.
- [227] Myungeun Eom and Byung-In Kim. The traffic signal control problem for intersections: a review. *European transport research review*, 12(1):1–20, 2020.

- [228] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2496–2505, 2018.
- [229] 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. *arXiv preprint arXiv:1509.02971*, 2015.
- [230] Khaled M Mohamed and AA Mohamad. A review of the development of hybrid atomistic–continuum methods for dense fluids. *Microfluidics and Nanofluidics*, 8(3):283–302, 2010.
- [231] Abhinav Golas, Rahul Narain, Jason Sewall, Pavel Krajcevski, Pradeep Dubey, and Ming Lin. Large-scale fluid simulation using velocity-vorticity domain decomposition. *ACM Transactions on Graphics (TOG)*, 31(6):1–9, 2012.
- [232] Rahul Narain, Abhinav Golas, and Ming C Lin. Free-flowing granular materials with two-way solid coupling. In *ACM SIGGRAPH Asia 2010 papers*, pages 1–10. 2010.
- [233] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. *ACM Transactions on Graphics (TOG)*, 25(3):1160–1168, 2006.
- [234] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C Lin. Aggregate dynamics for dense crowd simulation. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–8. 2009.
- [235] Yuanshi Zheng, Jingying Ma, and Long Wang. Consensus of hybrid multi-agent systems. *IEEE transactions on neural networks and learning systems*, 29(4):1359–1365, 2017.
- [236] Michael S Branicky, Vivek S Borkar, and Sanjoy K Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE transactions on automatic control*, 43(1):31–45, 1998.
- [237] Rafael Fierro, Aveek K Das, Vijay Kumar, and James P Ostrowski. Hybrid control of formations of robots. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 1, pages 157–162. IEEE, 2001.
- [238] Zheng Tang, Milind Naphade, Ming-Yu Liu, Xiaodong Yang, Stan Birchfield, Shuo Wang, Ratnesh Kumar, David Anastasiu, and Jenq-Neng Hwang. Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8797–8806, 2019.
- [239] Sanghyun Son, Yi-Ling Qiao, Jason Sewall, and Ming C Lin. Differentiable hybrid traffic simulation. *ACM Transactions on Graphics (TOG)*, 41(6):1–10, 2022.
- [240] S Krauss. Microscopic modeling of traffic flow: investigation of collision free vehicle dynamics, Apr 1998.
- [241] Adam Ścibior, Vasileios Lioutas, Daniele Reda, Peyman Bateni, and Frank Wood. Imagining the road ahead: Multi-agent trajectory prediction via differentiable simulation. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 720–725, 2021.

- [242] Laura Zheng, Sanghyun Son, and Ming C Lin. Traffic-aware autonomous driving with differentiable traffic simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3517–3523. IEEE, 2023.
- [243] Mehdi Rafati Fard, Afshin Shariat Mohaymany, and Matin Shahri. A new methodology for vehicle trajectory reconstruction based on wavelet analysis. *Transportation Research Part C: Emerging Technologies*, 74:150–167, 2017.
- [244] Aurélien Duret, Christine Buisson, and Nicolas Chiabaut. Estimating individual speed-spacing relationship and assessing ability of newell’s car-following model to reproduce trajectories. *Transportation research record*, 2088(1):188–197, 2008.
- [245] Saskia Ossen and Serge P Hoogendoorn. Validity of trajectory-based calibration approach of car-following models in presence of measurement errors. *Transportation Research Record*, 2088(1):117–125, 2008.
- [246] Christian Thiemann, Martin Treiber, and Arne Kesting. Estimating acceleration and lane-changing dynamics from next generation simulation trajectory data. *Transportation Research Record*, 2088(1):90–101, 2008.
- [247] Jungwook Jun, Randall Guensler, and Jennifer H Ogle. Smoothing methods to minimize impact of global positioning system random error on travel distance, speed, and acceleration profile estimates. *Transportation Research Record*, 1972(1):141–150, 2006.
- [248] Ari Seff, Brian Cera, Dian Chen, Mason Ng, Aurick Zhou, Nigamaa Nayakanti, Khaled S. Refaat, Rami Al-Rfou, and Benjamin Sapp. Motionlm: Multi-agent motion forecasting as language modeling, 2023.
- [249] Nigamaa Nayakanti, Rami Al-Rfou, Aurick Zhou, Kratarth Goel, Khaled S. Refaat, and Benjamin Sapp. Wayformer: Motion forecasting via simple & efficient attention networks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2980–2987, 2023.
- [250] Zikang Zhou, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Query-centric trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.
- [251] Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. Mtr++: Multi-agent motion prediction with symmetric scene modeling and guided intention querying. *arXiv preprint arXiv:2306.17770*, 2023.
- [252] Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. Motion transformer with global intention localization and local movement refinement. *Advances in Neural Information Processing Systems*, 2022.
- [253] Balakrishnan Varadarajan, Ahmed Hefny, Avikalp Srivastava, Khaled S. Refaat, Nigamaa Nayakanti, Andre Cornman, Kan Chen, Bertrand Douillard, Chi Pang Lam, Dragomir Anguelov, and Benjamin Sapp. Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction. In *2022 International Conference on Robotics and Automation (ICRA)*, page 7814–7821. IEEE Press, 2022.

- [254] Chen Shi, Shaoshuai Shi, and Li Jiang. Mtr v3: 1st place solution for 2024 waymo open dataset challenge - motion prediction, 2024.
- [255] Jiawei Sun, Chengran Yuan, Shuo Sun, Shanze Wang, Yuhang Han, Shuailei Ma, Zefan Huang, Anthony Wong, Keng Peng Tee, and Marcelo H Ang Jr. Controlmtr: Control-guided motion transformer with scene-compliant intention points for feasible motion prediction. *arXiv preprint arXiv:2404.10295*, 2024.
- [256] Scott Ettinger, Kratarth Goel, Avikalp Srivastava, and Rami Al-Rfou. Scaling motion forecasting models with ensemble distillation. *arXiv preprint arXiv:2404.03843*, 2024.
- [257] Vibha Bharilya and Neetesh Kumar. Machine learning for autonomous vehicle’s trajectory prediction: A comprehensive survey, challenges, and future research directions. *Vehicular Communications*, page 100733, 2024.
- [258] Vassili Alexiadis, James Colyar, John Halkias, Rob Hranac, and Gene McHale. The next generation simulation program. *Institute of Transportation Engineers. ITE Journal*, 74(8):22, 2004.
- [259] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.
- [260] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [261] Longzhong Lin, Xuewu Lin, Tianwei Lin, Lichao Huang, Rong Xiong, and Yue Wang. Eda: Evolving and distinct anchors for multimodal motion prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 3432–3440, 2024.
- [262] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018.
- [263] Benoit Guillard, Edoardo Remelli, Artem Lukoianov, Stephan R Richter, Timur Bagautdinov, Pierre Baque, and Pascal Fua. Deepmesh: Differentiable iso-surface extraction. *arXiv preprint arXiv:2106.11795*, 2021.
- [264] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (ToG)*, 38(4):1–12, 2019.
- [265] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 45–54, 2020.
- [266] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Company, 2013.

- [267] Jonathan W Brandt and V Ralph Algazi. Continuous skeleton computation by voronoi diagram. *CVGIP: Image understanding*, 55(3):329–338, 1992.
- [268] Nina Amenta, Marshall Bern, and David Eppstein. The crust and the β -skeleton: Combinatorial curve reconstruction. *Graphical models and image processing*, 60(2):125–135, 1998.
- [269] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421, 1998.
- [270] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.
- [271] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, 2008.
- [272] Clément Jamin, Sylvain Pion, and Monique Teillaud. 3D triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6 edition, 2023.
- [273] Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner. Meshgpt: Generating triangle meshes with decoder-only transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19615–19625, 2024.
- [274] Yiwen Chen, Tong He, Di Huang, Weicai Ye, Sijin Chen, Jiaxiang Tang, Xin Chen, Zhongang Cai, Lei Yang, Gang Yu, et al. Meshanything: Artist-created mesh generation with autoregressive transformers. *arXiv preprint arXiv:2406.10163*, 2024.
- [275] Yiwen Chen, Yikai Wang, Yihao Luo, Zhengyi Wang, Zilong Chen, Jun Zhu, Chi Zhang, and Guosheng Lin. Meshanything v2: Artist-created mesh generation with adjacent mesh tokenization. *arXiv preprint arXiv:2408.02555*, 2024.
- [276] Tianchang Shen, Zhaoshuo Li, Marc Law, Matan Atzmon, Sanja Fidler, James Lucas, Jun Gao, and Nicholas Sharp. Spacemesh: A continuous representation for learning manifold surface meshes. *arXiv preprint arXiv:2409.20562*, 2024.
- [277] Xinyue Wei, Kai Zhang, Sai Bi, Hao Tan, Fujun Luan, Valentin Deschaintre, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. Meshlrn: Large reconstruction model for high-quality meshes. *arXiv preprint arXiv:2404.12385*, 2024.
- [278] Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. Clay: A controllable large-scale generative model for creating high-quality 3d assets. *ACM Transactions on Graphics (TOG)*, 43(4):1–20, 2024.
- [279] Benoit Guillard, Federico Stella, and Pascal Fua. Meshudf: Fast and differentiable meshing of unsigned distance field networks. In *European conference on computer vision*, pages 576–592. Springer, 2022.

- [280] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023.
- [281] Franz Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.
- [282] Siu-Wing Cheng, Tamal Krishna Dey, Jonathan Shewchuk, and Sartaj Sahni. *Delaunay mesh generation*. CRC Press Boca Raton, 2013.
- [283] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020.
- [284] Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. *SIGGRAPH sketches*, 10(1):1, 2007.
- [285] Bui Tuong Phong. Illumination for computer generated pictures. In *Seminal graphics: pioneering efforts that shaped the field*, pages 95–101. 1998.
- [286] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 conference papers*, pages 1–11, 2024.
- [287] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.
- [288] Julian Chibane, Gerard Pons-Moll, et al. Neural unsigned distance fields for implicit function learning. *Advances in Neural Information Processing Systems*, 33:21638–21652, 2020.
- [289] Sanghyun Son, Matheus Gadelha, Yang Zhou, Matthew Fisher, Zexiang Xu, Yi-Ling Qiao, Ming C Lin, and Yi Zhou. Dmesh++: An efficient differentiable mesh for complex shapes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 26590–26599, 2025.
- [290] Gouri Dhatt, Emmanuel Lefrançois, and Gilbert Touzot. *Finite element method*. John Wiley & Sons, 2012.
- [291] Michal Křížek. On the maximum angle condition for linear tetrahedral elements. *SIAM Journal on Numerical Analysis*, 29(2):513–520, 1992.
- [292] Jonathan Richard Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In *11th International Meshing Roundtable, {IMR} 2002*, 2002.
- [293] Si Hang. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2):11, 2015.
- [294] Jean-Daniel Boissonnat, Olivier Devillers, Monique Teillaud, and Mariette Yvinec. Triangulations in cgal. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 11–18, 2000.

- [295] Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. Tetrahedral meshing in the wild. *ACM Trans. Graph.*, 37(4):60, 2018.
- [296] Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. Fast tetrahedral meshing in the wild. *ACM Transactions on Graphics (ToG)*, 39(4):117–1, 2020.
- [297] Mark A Yerry and Mark S Shephard. Automatic three-dimensional mesh generation by the modified-octree technique. *International journal for numerical methods in engineering*, 20(11):1965–1990, 1984.
- [298] Alexander Fuchs. Automatic grid generation with almost regular delaunay tetrahedra. In *IMR*, pages 133–147. Citeseer, 1998.
- [299] David J Naylor. Filling space with tetrahedra. *International Journal for Numerical Methods in Engineering*, 44(10):1383–1395, 1999.
- [300] François Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. In *ACM SIGGRAPH 2007 papers*, pages 57–es. 2007.
- [301] Minghao Guo, Bohan Wang, Kaiming He, and Wojciech Matusik. Tetsphere splatting: Representing high-quality geometry with lagrangian volumetric meshes. *arXiv preprint arXiv:2405.20283*, 2024.
- [302] Siu-Wing Cheng, Tamal K Dey, Herbert Edelsbrunner, Michael A Facello, and Shang-Hua Teng. Sliver exudation. *Journal of the ACM (JACM)*, 47(5):883–904, 2000.
- [303] Andreas Fabri and Sylvain Pion. Cgal: The computational geometry algorithms library. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 538–539, 2009.
- [304] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9601–9611, 2019.
- [305] Peng-Shuai Wang, Yang Liu, and Xin Tong. Dual octree graph networks for learning adaptive volumetric shape representations. *ACM Transactions on Graphics (TOG)*, 41(4):1–15, 2022.
- [306] Ashish Myles, Nico Pietroni, and Denis Zorin. Robust field-aligned global parametrization. *ACM Trans. Graph.*, 33(4):135–1, 2014.
- [307] Sanghyun Son, Laura Zheng, Ryan Sullivan, Yi-Ling Qiao, and Ming Lin. Gradient informed proximal policy optimization. *Advances in Neural Information Processing Systems*, 36:8788–8814, 2023.
- [308] Anh N Nhu, Sanghyun Son, and Ming Lin. Time-aware world model for adaptive prediction and control. In *Forty-second International Conference on Machine Learning*, 2025.

- [309] Sanghyun Son, Laura Zheng, Brian Clipp, Connor Greenwell, Sujin Philip, and Ming C Lin. Gradient-based trajectory optimization with parallelized differentiable traffic simulation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14497–14504. IEEE, 2025.
- [310] Denys Makoviichuk and Viktor Makoviyhuk. rl-games: A high-performance framework for reinforcement learning. https://github.com/Denys88/rl_games, May 2022.
- [311] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805–1824, Aug 2000.
- [312] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.
- [313] John Lee. *Introduction to topological manifolds*, volume 202. Springer Science & Business Media, 2010.