The ChainQueen Differentiable Physics Engine

by

Yuanming Hu

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

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2019

© Massachusetts Institute of Technology 2019. All rights reserved.

Author .....
Department of Electrical Engineering and Computer Science
November 16, 2018

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

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

The ChainQueen Differentiable Physics Engine

by

Yuanming Hu

Submitted to the Department of Electrical Engineering and Computer Science
on November 16, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Physical simulators have been widely used in robot planning and control. Among them, differentiable simulators are particularly favored, as they can be incorporated into gradient-based optimization algorithms that are efficient in solving inverse problems such as optimal control and motion planning. Simulating deformable objects is, however, more challenging compared to rigid body dynamics. The underlying physical laws of deformable objects are more complex, and the resulting systems have orders of magnitude more degrees of freedom and therefore they are significantly more computationally expensive to simulate. Computing gradients with respect to physical design or controller parameters is typically even more computationally challenging. In this paper, we propose a real-time, differentiable hybrid Lagrangian-Eulerian physical simulator for deformable objects, ChainQueen, based on the Moving Least Squares Material Point Method (MLS-MPM). MLS-MPM can simulate deformable objects including contact and can be seamlessly incorporated into inference, control and co-design systems. We demonstrate that our simulator achieves high precision in both forward simulation and backward gradient computation. We have successfully employed it in a diverse set of control tasks for soft robots, including problems with nearly $3,000$ decision variables.

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

Acknowledgments

First of all, I am grateful to my thesis advisor Prof. Wojciech Matusik, without whom this work would not have been made possible. His insightful guidance, generous support and kind encouragements have not only greatly accelerated the development of this project, but also cultivated my ability to conduct high-quality research.

Then, I would like to thank my collaborators. I feel privileged to have the opportunity to work with Prof. Bo Zhu, Prof. Eftychios Sifakis, Prof. Chenfanfu Jiang, Prof. William T. Freeman, Prof. Joshua B. Tenenbaum and Prof. Daniela Rus during my master study. Also, I am fortunate to collaborate with Jiancheng Liu, Andrew Spielberg, Tao Du, Jiajun Wu, Liang Shi, Haixiang Liu, Hao He, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Dr. Andre Pradhana, who have been both excellent colleagues and supportive friends of mine.

Finally and most importantly, I would thank my family and my girlfriend. I am indebted to them, since I have spent very limited time together with them in the past year, due to the physical distance across the oceans.

This thesis is dedicated to everyone above, and many others who have supported me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robot planning and control algorithms often rely on physical simulators for prediction and optimization [32, 6]. In particular, differentiable physical simulators enable the use of gradient-based optimizers, significantly improving control efficiency and precision. Motivated by this, there has been extensive research on differentiable rigid body simulators, using approximate [2, 24] and exact [5, 31, 8] methods.

Significant challenges remain for deformable objects. First, simulating the motion of deformable objects is slow, because they have much higher degrees of freedom (DoFs). Second, contact detection and resolution is challenging for deformable objects, due to their changing geometries and potential self-collisions. Third, closed-form and efficient computation of gradients is challenging in the presence of contact. As a consequence, current simulation methods for soft objects cannot be effectively used for solving inverse problems such as optimal control and motion planning.

In this work, we introduce a real-time, differentiable physical simulator for deformable objects, building upon the Moving Least Squares Material Point Method (MLS-MPM) [16]. We name our simulator ChainQueen*. The Material Point Method (MPM) is a hybrid Lagrangian-Eulerian method that uses both particles and grid nodes for simulation [29]. MLS-MPM accelerates and simplifies traditional MPM using a moving least squares force discretization. In ChainQueen, we introduce the first fully differentiable MLS-MPM simulator with respect to both state and model

*Or 乾坤 , literally "everything between the sky and the earth."

parameters, with both forward simulation and back-propagation running efficiently on GPUs. We demonstrate the ability to efficiently calculate gradients with respect to the entire simulation. This enables many novel applications for soft robotics including optimization-based closed-loop controller design, trajectory optimization, and co-design of robot geometry, materials, and control.

As a particle-grid-based hybrid simulator, MPM simulates objects of various states, such as liquid (e.g., water), granular materials (e.g., sand), and elastoplastic materials (e.g., snow and human tissue). ChainQueen focuses on elastic materials for soft robotics. It is fully differentiable and $4-9\times$ faster than the current state of the art. Numerical and experimental validation suggest that ChainQueen achieves high precision in both forward simulation and backward gradient computation.

ChainQueen's differentiability allows it to support gradient-based optimization for control and system identification. By performing gradient descent on controller parameters, our simulator is capable of solving these inverse problems on a diverse set of complex tasks, such as optimizing a 3D soft walker controller given an objective. Similarly, gradient descent on physical design parameters, enables inference of physical properties (e.g. mass, density and Young's modulus) of objects and optimizing design for a desired task.

In addition to benchmarking ChainQueen's performance and demonstrating its capabilities on a diverse set of inverse problems, we have interfaced our simulator with high-level python scripts to make ChainQueen user-friendly. Users at all levels will be able to develop their own soft robotics systems using our simulator, without the need to understand its low-level details. We will open-source our code and data and we hope they can benefit the robotics community.

Chapter 2

Related Work

2.1 Material Point Method

The material point method has been extensively developed from both a solid mechanics [29] and computer graphics [19] perspective. As a hybrid Eulerian-Langrangian method, MPM has demonstrated its versatility in simulating snow [28, 12], sand [21, 3], non-Newtonion fluids [25], cloth [18, 13], solid-fluid coupling [9, 30], rigid body coupling, and cutting [16]. [10] also proposed an adaptive MPM scheme to concentrate computation resources in the regions of interest.

There are many benefits of using MPM for soft robotics. First, MPM is a well-founded and physically-accurate discretization method and can be derived through the weak form of conservation laws. Such a physically based approach makes it easier to match simulation with real-world experiments. Second, MPM is friendly to parallelization on modern hardware architectures. Closely related to our work is a high-performance GPU implementation [11] by Gao et al., from which we borrow many useful optimization practices. Though efficient when solving forward simulation, their simulator is not differentiable, making it inefficient for inverse problems in robotics and learning. Third, MPM naturally handles large deformation and (self-)collision, which are common in soft robotics, but often not modeled in, e.g., mesh-based approaches due to computational expense. Finally, the continuum dynamics (including soft object collision) are governed by the smooth (and differentiable) potential energy, making

the whole system differentiable.

Our simulator, ChainQueen, is fully differentiable and the first simulator that applies MPM to soft robotics.

2.2 Differentiable Simulation and Control

Recently, there has been an increasing interest in building differentiable simulators for planning and control. For rigid bodies, [1], [2] and [24] proposed to approximate object interaction with neural nets; later, [26] explored their usage in control. Approximate analytic differentiable rigid body simulators have also been proposed [5, 4]. Such systems have been deployed for manipulation and planning [33].

Differentiable simulators for deformable objects have been less studied. Recently, [27] proposed SPNets for differentiable simulation of position-based fluids [22]. The particle interactions are coded as neural network operations and differentiability is achieved via automatic differentiation in PyTorch. A hierarchical particle-based object representation using neural networks is also proposed in [24]. Instead of approximating physics using neural networks, ChainQueen differentiates MLS-MPM, a well physically founded discretization scheme derived from continuum mechanics. In summary, our simulator can be used for a more diverse set of objects; it is more physically plausible, and runs faster.

Chapter 3

Background: The Moving Least Squares Material Point Method

In this chapter, we provide some background on the material point method (MPM), which is used for forward simulation.

We use the moving least squares material point method (MLS-MPM) [16] to discretize continuum mechanics, which is governed by the following two equations:

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g} \quad \text{(momentum conservation)}, \tag{3.1}$$

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v} = 0 \quad \text{(mass conservation)}. \tag{3.2}$$

MLS-MPM is a simpler and more efficient variant of classical MPM. The simplicity of MLS-MPM has greatly reduced the work needed to differentiate it. Here we briefly cover the basics of MLS-MPM and readers are referred to [19] and [16] for a comprehensive introduction of MPM and MLS-MPM, respectively.

The material point method is a hybrid Eulerian-Lagrangian method, where both particles and grid nodes are used. Simulation state information is transferred back-and-forth between these two representations. We summarize the notations we use in this paper in Table 3.1. Subscripts are used to denote particle (p) and grid nodes (i), while superscripts (n, $n+1$) are used to distinguish quantities in different time steps.

The MLS-MPM simulation cycle has three steps:

Table 3.1: List of notations for MLS-MPM.

Symbol	Type	Affiliation	Meaning
Δt	scalar		time step size
Δx	scalar		grid cell size
$\mathbf{x}_p$	vector	particle	position
V_p^0	scalar	particle	initial volume
$\mathbf{v}_p$	vector	particle	velocity
$\mathbf{C}_p$	matrix	particle	affine velocity field [17]
$\mathbf{P}_p$	matrix	particle	PK1 stress ($\partial\psi_p/\partial\mathbf{F}_p$)
$\boldsymbol{\sigma}_{pa}$	matrix	particle	actuation Cauchy stress
$\mathbf{A}_p$	matrix	particle	actuation stress (material space)
$\mathbf{F}_p$	matrix	particle	deformation gradient
$\mathbf{x}_i$	vector	node	position
m_i	scalar	node	mass
$\mathbf{v}_i$	vector	node	velocity
$\mathbf{p}_i$	vector	node	momentum, i.e. $m_i\mathbf{v}_i$
N	scalar		quadratic B-spline function

1. **Particle-to-grid transfer (P2G).** Particles transfer mass m_p, momentum $(m\mathbf{v})_p^n$, and stress-contributed impulse to their neighbouring grid nodes, using the Affine Particle-in-Cell method (APIC) [17] and moving least squares force discretization [16], weighted by a compact B-spline kernel N:

$$m_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) m_p, \tag{3.3}$$

$$\mathbf{G}_p^n = -\frac{4}{\Delta x^2}\Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n, \tag{3.4}$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)\left[m_p \mathbf{v}_p^n + \mathbf{G}_p^n(\mathbf{x}_i - \mathbf{x}_p^n)\right]. \tag{3.5}$$

2. **Grid operations.** Grid momentum is normalized into grid velocity after division by grid mass:

$$\mathbf{v}_i^n = \frac{1}{m_i^n}\mathbf{p}_i^n. \tag{3.6}$$

Note that neighbouring particles interact with each other through their shared grid nodes, and collisions are handled automatically. Here we omit boundary

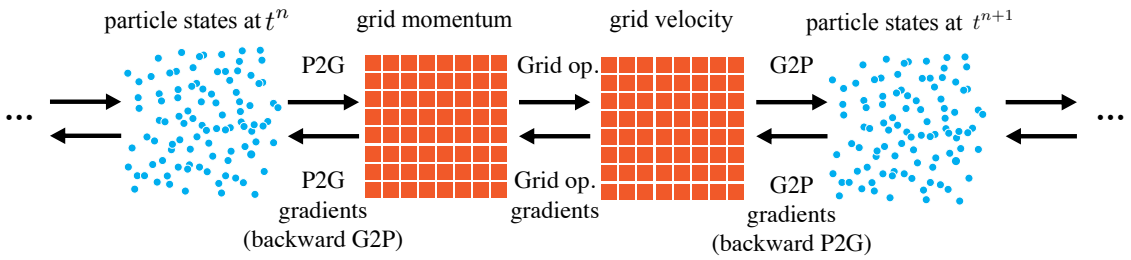


Figure 3-1: One time step of MLS-MPM. Top arrows are for forward simulation and bottom ones are for back propagation. A controller is embedded in the P2G process to generate actuation given particle configurations.

conditions and gravity for simplicity. Boundary conditions are usually applied as a projection on grid node velocity.

3. **Grid-to-particle transfer (G2P).** Particles gather updated velocity $\mathbf{v}_p^{n+1}$, local velocity field gradients $\mathbf{C}_p^{n+1}$ and position $\mathbf{x}_p^{n+1}$ form neighbouring grid nodes. At the same time, the constitutive model properties (for example, deformation gradients $\mathbf{F}_p^{n+1}$) are updated. MLS-MPM unifies the local velocity gradient computation and affine velocity field computation, leading to higher efficiency and simplicity.

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n, \tag{3.7}$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2}\sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n(\mathbf{x}_i - \mathbf{x}_p^n)^T, \tag{3.8}$$

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t\mathbf{C}_p^{n+1})\mathbf{F}_p^n, \tag{3.9}$$

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t\mathbf{v}_p^{n+1}. \tag{3.10}$$

For soft robotics, we additionally introduce an actuation model. Inspired by actuators such as [14], we designed an actuation model that expands or stretches particle p via an additional Cauchy stress $\mathbf{A}_p = \mathbf{F}_p\boldsymbol{\sigma}_{pa}\mathbf{F}_p^T$, with $\boldsymbol{\sigma}_{pa} = \text{Diag}(a_x, a_y, a_z)$ – the stress in the material space. This framework supports the use of other actuation models including pneumatic, hydraulic, and cable-driven actuators. Fig. 3-1 illustrates

forward simulation, and back propagation, which we will introduce in the next chapter.

Chapter 4

The Differentiable Moving Least Squares Material Point Method

MLS-MPM is naturally differentiable. Although the forward MPM simulation has been extensively used in computer graphics, the backward direction (differentiation or back-propagation) is largely unexplored. The backward gradients can be derived given forward simulation relationships. Here we show the gradients of $\mathbf{x}_p^n$ based on

forward simulation, as an example:

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \tag{4.1}$$

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n \tag{4.2}$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2}\sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n(\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{4.3}$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)\left[m_p\mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2}\Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n\right)(\mathbf{x}_i - \mathbf{x}_p^n)\right] \tag{4.4}$$

$$m_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) m_p \tag{4.5}$$

$$\mathbf{G}_p := \left(-\frac{4}{\Delta x^2}V_p^0 \Delta t \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n\right) \tag{4.6}$$

$$\Longrightarrow \tag{4.7}$$

$$\frac{\partial L}{\partial \mathbf{x}_{p\alpha}^n} = \left[\frac{\partial L}{\partial \mathbf{x}_p^{n+1}}\frac{\partial \mathbf{x}_p^{n+1}}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial \mathbf{v}_p^{n+1}}\frac{\partial \mathbf{v}_p^{n+1}}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial \mathbf{C}_p^{n+1}}\frac{\partial \mathbf{C}_p^{n+1}}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial \mathbf{p}_i^n}\frac{\partial \mathbf{p}_i^n}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial m_i^n}\frac{\partial m_i^n}{\partial \mathbf{x}_p^n}\right]_\alpha \tag{4.8}$$

$$= \frac{\partial L}{\partial \mathbf{x}_{p\alpha}^{n+1}} \tag{4.9}$$

$$+ \sum_i \sum_\beta \frac{\partial L}{\partial \mathbf{v}_{p\beta}^{n+1}}\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}}\mathbf{v}_{i\beta}^n \tag{4.10}$$

$$+ \sum_i \sum_\beta \tag{4.11}$$

$$\frac{4}{\Delta x^2}\left\{-\frac{\partial L}{\partial \mathbf{C}_{p\beta\alpha}^{n+1}}N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_{i\beta} + \sum_\gamma \frac{\partial L}{\partial \mathbf{C}_{p\beta\gamma}^{n+1}}\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}}\mathbf{v}_{i\beta}(\mathbf{x}_{i\gamma} - \mathbf{x}_{p\gamma})\right\} \tag{4.12}$$

$$+ \sum_i \sum_\beta \tag{4.13}$$

$$\frac{\partial L}{\partial \mathbf{p}_{i\beta}^n}\left[\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}}\left(m_p\mathbf{v}_{p\beta}^n + [\mathbf{G}_p(\mathbf{x}_i - \mathbf{x}_p^n)]_\beta\right) - N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{G}_{p\beta\alpha}\right] \tag{4.14}$$

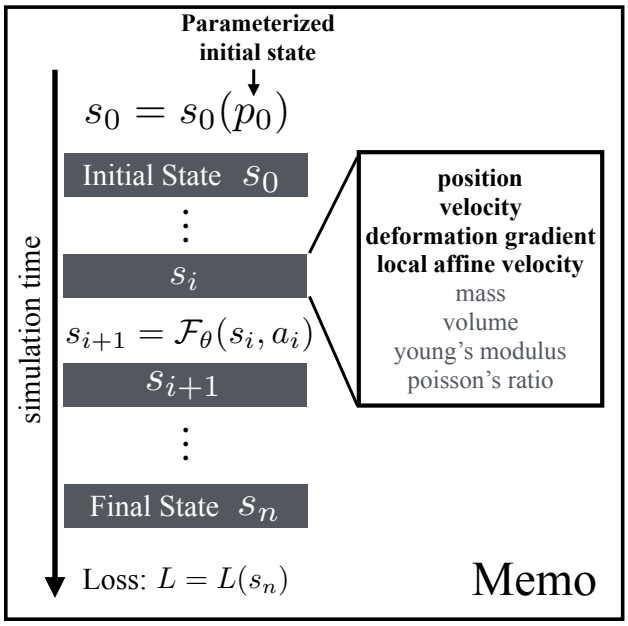
$$+ m_p \sum_i \frac{\partial L}{\partial m_i^n}\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}} \tag{4.15}$$

We refer the readers to Appendix A for more details on gradient derivation. Based on the gradients we have derived analytically, we have designed a high-performance implementation that resembles the traditional forward MPM cycle: backward P2G (scatter particle gradients to grid), grid operations, and backward G2P (gather grid gradients to particles). Gradients of state at the end of a time step with respect to states at the starting of the time step can be computed using the chain rule. With the single-step gradients computed, applying the chain rule at a higher level from the final state all-the-way to the initial state yields gradients of the final state with

respect to the initial state, as well as to the controller parameters that are used in each state. We cache all the simulation states in memory, using a "memo" object. Though the underlying differentiation is complicated, we have designed a simple high-level TensorFlow interface on which end-users can build their applications (Fig. 4-1).

Our high-performance implementation* takes advantage of the computational power of modern GPUs through CUDA. We also implemented a reference implementation in TensorFlow. Note that programming physical simulation as a "computation graph" using high-level frameworks such as TensorFlow is less inefficient. In fact, when all the overheads are gone, our optimized CUDA solver is 132× faster than the TensorFlow reference version. This is because TensorFlow is optimized towards deep learning applications where data granularity is much larger and memory access pattern is much more regular than physical simulation, and limited CPU-GPU bandwidth. In contrast, our CUDA implementation is tailored for MLS-MPM and explicitly optimized for parallelism and locality, thus delivering high-performance.

*Based on the **Taichi** [15] open source computer graphics library.




```
sym = sim.gradients_sym(loss,
                        variables=trainables)
```

Analytical gradients (CUDA)

$$\frac{\partial L}{\partial p_0} \quad \frac{\partial L}{\partial \theta}$$

```
memo = sim.run(
       initial_state=initial_state,
       num_steps=100,
       iteration_feed_dict={goal: goal_input},
       loss=loss)

grad = sim.eval_gradients(sym=sym, memo=memo)
gradient_descent = [
    v.assign(v - lr * g)
      for v, g in zip(trainables, grad)
]
sess.run(gradient_descent)
```

Figure 4-1: Left: A "memo" object consists all information of a single simulation execution, including all the time step state information (position, velocity, deformation gradients etc.), and parameters for the initial state p_0, policy parameter θ. Right: Code samples to get the symbolic differentiation (top) and memo, evaluate gradients out of the memo and symbolic differentiation, and finally use them for gradient descent (bottom).

Chapter 5

Evaluation

In this section, we conduct a comprehensive study of the efficiency and accuracy of our system, in both 2D and 3D.

5.1 Efficiency

Instead of using complex geometries, a simple falling cube is used for performance benchmarking, to ensure easy analysis and reproducibility. We benchmark the performance of our CUDA simulator against NVIDIA Flex [23], a popular PBD physical simulator capable of simulating deformable objects. Note that both PBD and MLS-MPM needs substepping iterations to ensure high stiffness. To ensure fair comparison, we set a Young's modulus, Poisson's ration and density so that visually ChainQueen gives similar results to Flex. We used two steps per frame and four iterations per step in Flex. Note that setting exactly the same parameters is not possible since in PBD there is no explicitly defined physical quantity such as Young's modulus.

We summarize the quantitative performance in Table 5.1. Our CUDA simulator provides higher speed than Flex, when the number of particles are the same. It is also worth noting that the TensorFlow implementation is much slower, due to excessive runtime overheads.

Table 5.1: Performance comparisons on a NVIDIA GTX 1080 Ti GPU. **F** stands for forward simulation and **B** stands for backward differentiation. **TF** indicates the TensorFlow implementation. When benchmarking our simulator with CUDA we use the C++ instead of python interface to avoid the extra overhead due to the TensorFlow runtime library.

Approach	Impl.	# Particles	Time per Frame
Flex (3D)	CUDA	8,024	3.5 ms (286 FPS)
Ours (3D, F)	CUDA	8,000	0.392 ms (**2,551** FPS)
Ours (3D, B)	CUDA	8,000	0.406 ms (**2,463** FPS)
Flex (3D)	CUDA	61,238	6 ms (167 FPS)
Ours (3D, F)	CUDA	64,000	1.594 ms (**628** FPS)
Ours (3D, B)	CUDA	64,000	1.774 ms (**563** FPS)
Ours (3D, F)	CUDA	512,000	10.501 ms (**92** FPS)
Ours (3D, B)	CUDA	512,000	11.594 ms (**86** FPS)
Ours (2D, F)	TF	6,400	13.2 ms (76 FPS)
Ours (2D, B)	TF	6,400	35.7 ms (28 FPS)
Ours (2D, F)	CUDA	6,400	0.10 ms (**10,000 FPS**)
Ours (2D, B)	CUDA	6,400	0.14 ms (**7,162 FPS**)

5.2 Accuracy

We design five test cases to evaluate the accuracy of both forward simulation and backward gradient evaluation:

1. A1 (analytic, 3D, float32 precision): final position w.r.t. initial velocity (with collision). This case tests conservation of momentum, gradient accuracy and stability of back-propagation.

2. A2 (analytic, 3D, float32 precision): same as A1 but with one collision to a friction-less wall.

3. B (numeric, 2D, float64 precision): colliding billiards. This case tests gradient accuracy and stability in more complex cases where analytic solutions do not exist. We used float64 precision for accurate finite difference results.

4. C (numeric, 2D, float64 precision): finger controller. This case tests gradient accuracy of controller parameters, which are used repeatedly in the whole

Table 5.2: Relative error in simulation and gradient precision. Empty values are because of too short time for collision to happen.

Case	1 steps	10 steps	100 steps	1000 steps
A1	9.80×10^{-8}	4.74×10^{-8}	1.15×10^{-7}	1.43×10^{-5}
A2	-	-	-	2.69×10^{-5}
B	-	-	2.39×10^{-8}	2.83×10^{-8}
C	5.63×10^{-6}	2.24×10^{-7}	6.97×10^{-7}	1.76×10^{-6}

simulation process.

5. D1 (experimental, pneumatic actuator, actuation) In order to evaluate our simulator's real-world accuracy, we compared the deformation of a physical actuator to a virtual one. The physical actuator has four pneumatic chambers which can be inflated with an external pump, arranged in a cross-shape. Inflating the individual chambers bends the actuator away from that chamber. The actuator was cast using Smooth-On Dragon Skin 30.

6. D2 (experimental, pneumatic actuator, bouncing) In a second test, we dropped the same actuator from a 15 cm height, and compared its dynamic motion to a simulation.

In 3D analytic test cases, where gradients w.r.t. initial velocity can be directly evaluated as in Table 5.2. For the experimental comparisons, the results are shown in Fig. 5-1. In addition to our simulator's high performance and accuracy, it is worth noting that that the gradients remain stable in the long term, within up to 1000 time steps.

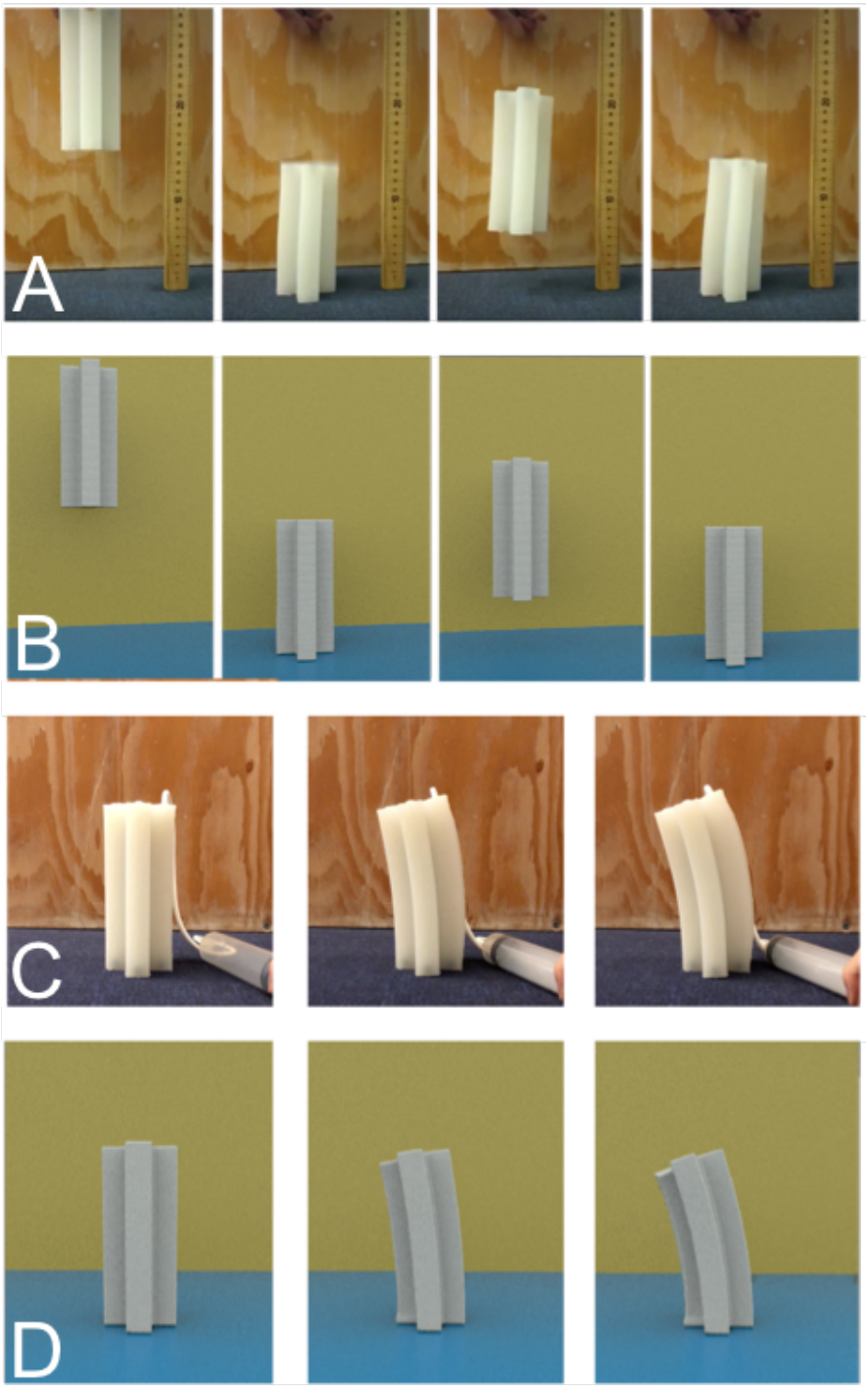


Figure 5-1: Experiments on the pneumatic leg. **Row (A, B)** Footage and simulator results of a bouncing experiment with the leg dropping at 15 cm. **Row (C, D)** Actuation test.

Chapter 6

Applications

The most attractive feature of our simulator is the existence of quickly computable gradients, which allows the use of much more efficient gradient-based optimization algorithms. In this section, we show the effectiveness of our differentiable simulator on gradient-based optimization tasks, including physical inference, control for soft robotics, and co-design of robotic arms.

6.1 Physical Parameter Inference

ChainQueen can be used to infer physical system properties given its observed motion, e.g. perform gradient descent to infer the relative densities of two colliding elastic balls (see Figure 6-1, ball A moving to the right hitting ball B, and ball B arrives the destination C). Gradient-based optimization infers that relative density of ball A is 2.26, which constitutes to the correct momentum to push B to C. Such capability makes it useful for real-world robotic tasks such as system identification.

6.2 Control

We can optimize regression-based controllers for soft robots and efficiently discover stable gaits. The controller takes as input the state vector $\mathbf{z}$, which includes target position, the center of mass position, and velocity of each composed soft component.

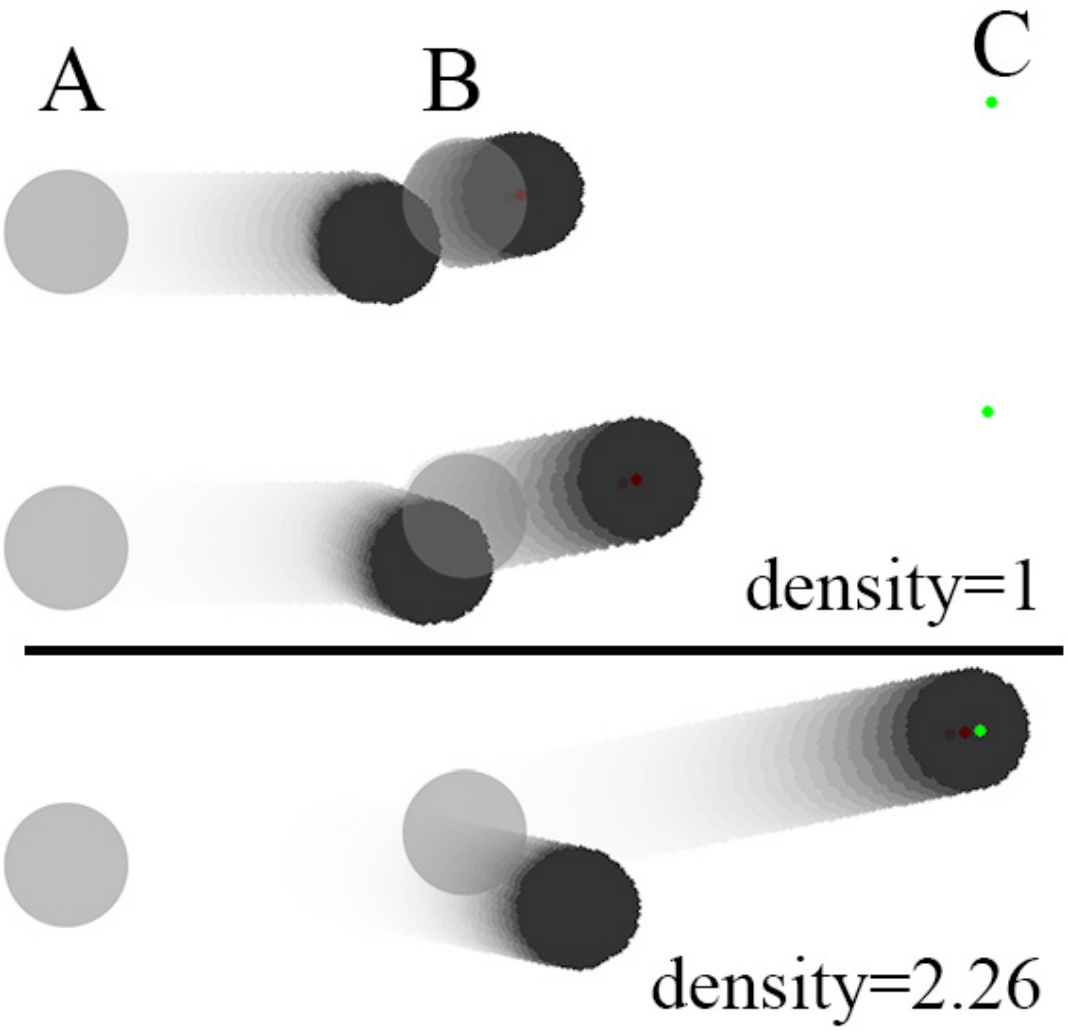


Figure 6-1: System Identification. The initial density 1 for elastic ball A leads to limited momentum to push ball B to its destination C. After several gradient descent iterations, the inferenced (optimized) density 2.26 gives the right amount of momentum for ball B to stop at C at the end of the simulation.

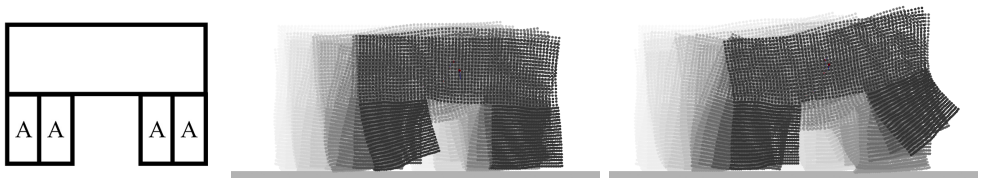


Figure 6-2: A soft 2D walker with controller optimized using gradient descent, aiming to achieve a maximum distance after 600 simulation steps. The walker has four actuators (left, marked by letter 'A's) with each capable of stretching or compressing in the vertical direction.

In our examples, the actuation vector $\mathbf{a}$ for up to 16 actuators is generated by the controller in each time step. During optimization, we perform gradient descent on variables $\mathbf{W}$ and $\mathbf{b}$, where $\mathbf{a} = \tanh(\mathbf{W}\mathbf{z} + \mathbf{b})$ is the actuation-generating controller.

We have designed a series of experiments (Fig. 6-2 and Fig. 6-3). Gradient-based optimizers successfully compute desired closed loop controllers controllers within only tens or hundreds of iterations.

6.3 Co-design

Our simulator is capable of not only providing gradients with respect to dynamics and controller parameters, but also with respect to structural design parameters, enabling co-design of soft robots. To demonstrate this, we designed a multi-link robot arm (two links, two joints each with two side-by-side actuators; all parts deformable). Similar to shooting method trajectory optimization, actuation for each time step is solved for, along with the time-invariant Young's modulus of the system for each particle. In our task, we optimized the end-effector of the arm to reach a goal ball with final 0 arm velocity, and minimized for actuation cost $\sum_{i=0}^{N} u_i^T u_i dt$, where u_i is the actuation vector at timestep i, and N is the total number of timesteps. This is a *dynamic* task and the target pose cannot be reached in a static equilibrium. NLOPT's sequential least squares programming algorithm was used for optimization [20]. We compared our co-design solution to fixed designs. The designed stiffness distribution is shown in Fig. 6-4, along with controls. The convergence for the different tasks can be seen in Fig. 6-5. As can be seen, only the co-design arm fully converges to the target goal, and with lower actuation cost. Actuation for each chamber was clamped, and rnges

of 30% to 400% of a dimensionless initial Young's modulus were allowed and chosen large enough such as to require a swing instead of a simple bend.

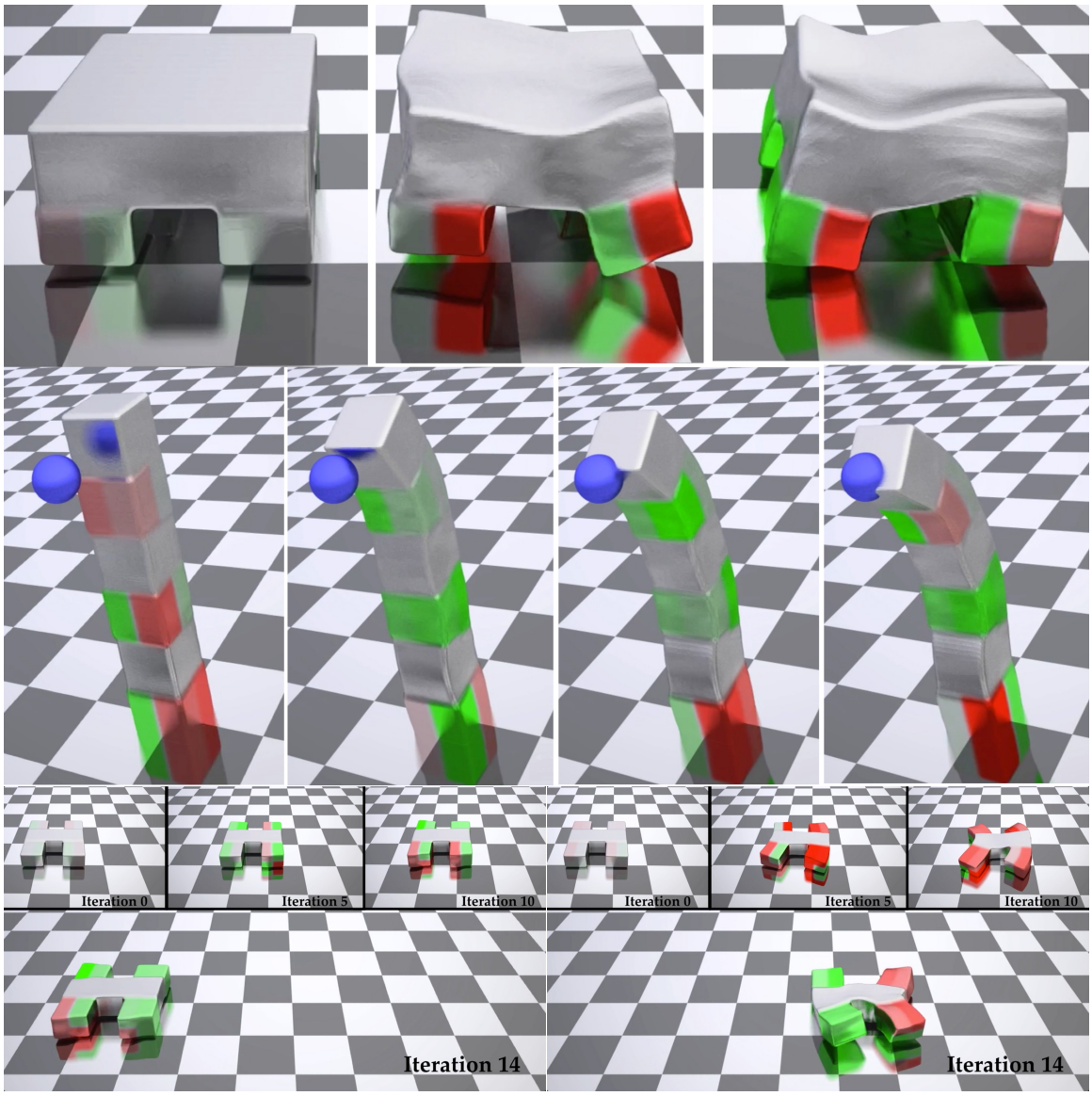
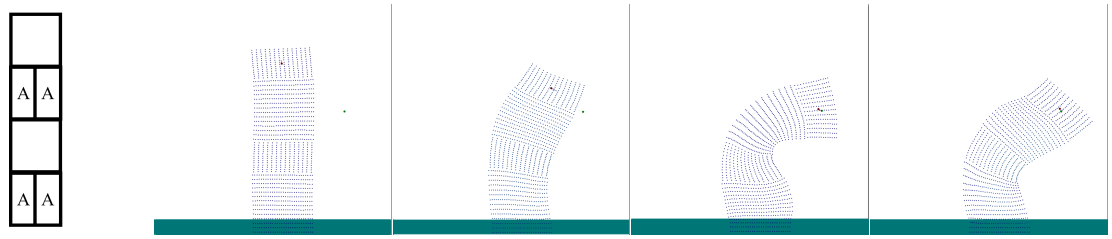


Figure 6-3: 3D robots with optimized controllers: quadruped runner, robotic finger, and crawler. The crawler is optimized with an open-loop controller, taking only trigonometric phase functions of time t as inputs.



(a) Actuation config (b) Resting pose (c) Final pose I (d) Final pose II (e) Final pose III

Figure 6-4: **Final poses of the arm swing task.** Lighter colors refer to stiffer regions. (c) Final pose of the fixed-stiffness 300% initial Young's modulus arm. (d) Final pose of the fixed-stiffness 300% initial Young's modulus arm. (e) Final pose of the co-optimized arm. Actuation cost is 95.5% that of the fixed 100% initial Young's modulus arm and converges. Only the co-optimized arm is able to fully reach its target. The final optimized spatially varying stiffness of the arm has lower stiffness on the outside of the bend, and higher stiffness inside, promoting more bend to the left. Qualitatively, this is similar in effect to the pleating on soft robot fingers.

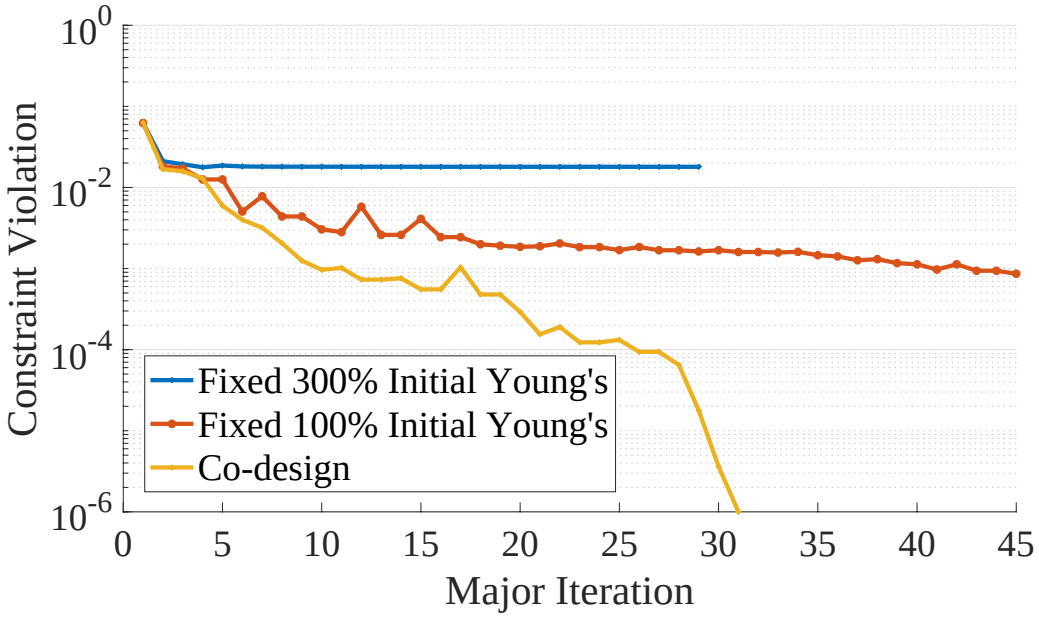


Figure 6-5: Convergence of the arm reaching task for co-design vs. fixed arm designs. The fixed designs can make progress but not complete the task, while with co-design, the task can be completed and the actuation cost is lower. Constraint violation is the norm of two constraints: distance of end-effector to goal and mean squared velocity of the particles.

Chapter 7

Discussion

We have presented ChainQueen, a differentiable simulator for soft robotics, and demonstrated how it can be deployed for inference, control, and co-design. ChainQueen has the potential to accelerate the development of soft robots. We have also developed a high-performance GPU implementation for ChainQueen, which we will open source.

One interesting future direction is to couple our soft object simulation with rigid body simulation, as done in [16]. As derived in [7], the Δt limit for explicit time integration is $C\Delta x\sqrt{\frac{\rho}{E}}$, where C is a constant close to one, ρ is the density, and E is the Young's modulus. That means for very stiff materials (e.g., rigid bodies), only a very restrictive Δt can be used. However, a rigid body simulator should probably be employed in the realm of nearly-rigid objects and coupled with our deformable body simulator. Combining our simulator with existing rigid-body simulators using Compatible Particle-in-Cell [16] can be an interesting direction.

Appendix A

Differentiation

In this appendix, we discuss the detailed steps for backward gradient computation in ChainQueen, i.e. the differentiable Moving Least Squares Material Point Method (MLS-MPM) [16]. Again, we summarize the notations in Table A.1. We assume fixed particle mass m_p, volume V_p^0, hyperelastic constitutive model (with potential energy ψ_p or Young's modulus E_p and Poisson's ratio ν_p) for simplicity.

A.1 Variable dependencies

The MLS-MPM time stepping is defined as follows:

Table A.1: List of notations for MLS-MPM.

Symbol	Type	Affiliation	Meaning
Δt	scalar		time step size
Δx	scalar		grid cell size
$\mathbf{x}_p$	vector	particle	position
V_p^0	scalar	particle	initial volume
$\mathbf{v}_p$	vector	particle	velocity
$\mathbf{C}_p$	matrix	particle	affine velocity field [17]
$\mathbf{P}_p$	matrix	particle	PK1 stress ($\partial\psi_p/\partial\mathbf{F}_p$)
$\boldsymbol{\sigma}_{pa}$	matrix	particle	actuation Cauchy stress
$\mathbf{A}_p$	matrix	particle	actuation stress (material space)
$\mathbf{F}_p$	matrix	particle	deformation gradient
$\mathbf{x}_i$	vector	node	position
m_i	scalar	node	mass
$\mathbf{v}_i$	vector	node	velocity
$\mathbf{p}_i$	vector	node	momentum, i.e. $m_i\mathbf{v}_i$
N	scalar		quadratic B-spline function

$$\mathbf{P}_p^n = \mathbf{P}_p^n(\mathbf{F}_p^n) + \mathbf{F}_p\boldsymbol{\sigma}_{pa}^n \tag{A.1}$$

$$m_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)m_p \tag{A.2}$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)\left[m_p\mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2}\Delta t V_p^0\mathbf{P}_p^n\mathbf{F}_p^{nT} + m_p\mathbf{C}_p^n\right)(\mathbf{x}_i - \mathbf{x}_p^n)\right] \tag{A.3}$$

$$\mathbf{v}_i^n = \frac{1}{m_i^n}\mathbf{p}_i^n \tag{A.4}$$

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n \tag{A.5}$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2}\sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n(\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{A.6}$$

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t\mathbf{C}_p^{n+1})\mathbf{F}_p^n, \tag{A.7}$$

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t\mathbf{v}_p^{n+1} \tag{A.8}$$

The forward variable dependency is as follows:

$$\mathbf{x}_p^{n+1} \leftarrow \mathbf{x}_p^n, \mathbf{v}_p^{n+1} \tag{A.9}$$

$$\mathbf{v}_p^{n+1} \leftarrow \mathbf{x}_p^n, \mathbf{v}_i^n \tag{A.10}$$

$$\mathbf{C}_p^{n+1} \leftarrow \mathbf{x}_p^n, \mathbf{v}_i^n \tag{A.11}$$

$$\mathbf{F}_p^{n+1} \leftarrow \mathbf{F}_p^n, \mathbf{C}_p^{n+1} \tag{A.12}$$

$$\mathbf{p}_i^n \leftarrow \mathbf{x}_p^n, \mathbf{C}_p^n, \mathbf{v}_p^n, \mathbf{P}_p^n, \mathbf{F}_p^n \tag{A.13}$$

$$\mathbf{v}_i^n \leftarrow \mathbf{p}_i^n, m_i^n \tag{A.14}$$

$$\mathbf{P}_p^n \leftarrow \mathbf{F}_p^n, \boldsymbol{\sigma}_{pa}^n \tag{A.15}$$

$$m_i^n \leftarrow \mathbf{x}_p^n \tag{A.16}$$

During back-propagation, we have the following reversed variable dependency:

$$\mathbf{x}_p^{n+1}, \mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1}, \mathbf{p}_i^{n+1}, m_i \leftarrow \mathbf{x}_p^n \tag{A.17}$$

$$\mathbf{p}_i^n \leftarrow \mathbf{v}_p^n \tag{A.18}$$

$$\mathbf{x}_p^{n+1} \leftarrow \mathbf{v}_p^{n+1} \tag{A.19}$$

$$\mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1} \leftarrow \mathbf{v}_i^n \tag{A.20}$$

$$\mathbf{F}_p^{n+1}, \mathbf{P}_p^n, \mathbf{p}_i^n \leftarrow \mathbf{F}_p^n \tag{A.21}$$

$$\mathbf{F}_p^{n+1} \leftarrow \mathbf{C}_p^{n+1} \tag{A.22}$$

$$\mathbf{p}_i^n \leftarrow \mathbf{C}_p^n \tag{A.23}$$

$$\mathbf{v}_i^n \leftarrow \mathbf{p}_i^n \tag{A.24}$$

$$\mathbf{v}_i^n \leftarrow m_i^n \tag{A.25}$$

$$\mathbf{p}_i^n \leftarrow \mathbf{P}_p^n \tag{A.26}$$

$$\mathbf{P}_p^n \leftarrow \boldsymbol{\sigma}_{pa}^n \tag{A.27}$$

We reverse swap two sides of the equations for easier differentiation derivation:

$$\mathbf{x}_p^n \rightarrow \mathbf{x}_p^{n+1}, \mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1}, \mathbf{p}_i^{n+1}, m_i \tag{A.28}$$

$$\mathbf{v}_p^n \rightarrow \mathbf{p}_i^n \tag{A.29}$$

$$\mathbf{v}_p^{n+1} \rightarrow \mathbf{x}_p^{n+1} \tag{A.30}$$

$$\mathbf{v}_i^n \rightarrow \mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1} \tag{A.31}$$

$$\mathbf{F}_p^n \rightarrow \mathbf{F}_p^{n+1}, \mathbf{P}_p^n, \mathbf{p}_i^n \tag{A.32}$$

$$\mathbf{C}_p^{n+1} \rightarrow \mathbf{F}_p^{n+1} \tag{A.33}$$

$$\mathbf{C}_p^n \rightarrow \mathbf{p}_i^n \tag{A.34}$$

$$\mathbf{p}_i^n \rightarrow \mathbf{v}_i^n \tag{A.35}$$

$$m_i^n \rightarrow \mathbf{v}_i^n \tag{A.36}$$

$$\mathbf{P}_p^n \rightarrow \mathbf{p}_i^n \tag{A.37}$$

$$\boldsymbol{\sigma}_{pa}^n \rightarrow \mathbf{P}_p^n \tag{A.38}$$

In the following sections, we derive detailed gradient relationships, in the order of actual gradient computation. The frictional boundary condition gradients are postponed to the end since it is less central, though during computation it belongs to grid operations. Back-propagation in ChainQueen is essentially a reversed process of forward simulation. The computation has three steps, backward particle to grid (P2G), backward grid operations, and backward grid to particle (G2P).

A.2 Backward Particle to Grid (P2G)

(A, P2G) For $\mathbf{v}_p^{n+1}$, we have

$$\begin{aligned} \mathbf{x}_p^{n+1} &= \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} && \text{(A.39)} \\ \Longrightarrow \frac{\partial L}{\partial \mathbf{v}_{p\alpha}^{n+1}} &= \left[\frac{\partial L}{\partial \mathbf{x}_p^{n+1}} \frac{\partial \mathbf{x}_p^{n+1}}{\partial \mathbf{v}_p^{n+1}}\right]_\alpha && \text{(A.40)} \\ &= \Delta t \frac{\partial L}{\partial \mathbf{x}_{p\alpha}^{n+1}}. && \text{(A.41)} \end{aligned}$$

(B, P2G) For $\mathbf{C}_p^{n+1}$, we have

$$\begin{aligned} \mathbf{F}_p^{n+1} &= (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1}) \mathbf{F}_p^n && \text{(A.42)} \\ \Longrightarrow \frac{\partial L}{\partial \mathbf{C}_{p\alpha\beta}^{n+1}} &= \left[\frac{\partial L}{\partial \mathbf{F}_p^{n+1}} \frac{\partial \mathbf{F}_p^{n+1}}{\partial \mathbf{C}_p^{n+1}}\right]_{\alpha\beta} && \text{(A.43)} \\ &= \Delta t \sum_\gamma \frac{\partial L}{\partial \mathbf{F}_{p\alpha\gamma}^{n+1}} \mathbf{F}_{p\beta\gamma}^n. && \text{(A.44)} \end{aligned}$$

Note, the above two gradients should also include the contributions of $\frac{\partial L}{\partial \mathbf{v}_p^n}$ and $\frac{\partial L}{\partial \mathbf{C}_p^n}$ respectively, with n being the next time step.

(C, P2G) For $\mathbf{v}_i^n$, we have

$$\begin{aligned} \mathbf{v}_p^{n+1} &= \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n && \text{(A.45)} \\ \mathbf{C}_p^{n+1} &= \frac{4}{\Delta x^2} \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n (\mathbf{x}_i - \mathbf{x}_p^n)^T && \text{(A.46)} \\ \Longrightarrow \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^n} &= \left[\sum_p \frac{\partial L}{\partial \mathbf{v}_p^{n+1}} \frac{\partial \mathbf{v}_p^{n+1}}{\partial \mathbf{v}_i^n} + \sum_p \frac{\partial L}{\partial \mathbf{C}_p^{n+1}} \frac{\partial \mathbf{C}_p^{n+1}}{\partial \mathbf{v}_i^n}\right]_\alpha && \text{(A.47)} \\ &= \sum_p \left[\frac{\partial L}{\partial \mathbf{v}_{p\alpha}^{n+1}} N(\mathbf{x}_i - \mathbf{x}_p^n) + \frac{4}{\Delta x^2} N(\mathbf{x}_i - \mathbf{x}_p^n) \sum_\beta \frac{\partial L}{\partial \mathbf{C}_{p\alpha\beta}^{n+1}} (\mathbf{x}_{i\beta} - \mathbf{x}_{p\beta})\right]. && \text{(A.48)} \end{aligned}$$

A.3 Backward Grid Operations

(D, grid) For $\mathbf{p}_i^n$, we have

$$\begin{aligned} \mathbf{v}_i^n &= \frac{1}{m_i^n}\mathbf{p}_i^n && \text{(A.49)} \\ \Longrightarrow \frac{\partial L}{\partial \mathbf{p}_{i\alpha}^n} &= \left[\frac{\partial L}{\partial \mathbf{v}_i^n}\frac{\partial \mathbf{v}_i^n}{\partial \mathbf{p}_i^n}\right]_\alpha && \text{(A.50)} \\ &= \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^n}\frac{1}{m_i^n}. && \text{(A.51)} \end{aligned}$$

(E, grid) For m_i^n, we have

$$\begin{aligned} \mathbf{v}_i^n &= \frac{1}{m_i^n}\mathbf{p}_i^n && \text{(A.52)} \\ \Longrightarrow \frac{\partial L}{\partial m_i^n} &= \frac{\partial L}{\partial \mathbf{v}_i^n}\frac{\partial \mathbf{v}_i^n}{\partial m_i^n} && \text{(A.53)} \\ &= -\frac{1}{(m_i^n)^2}\sum_\alpha \mathbf{p}_{i\alpha}^n \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^n} && \text{(A.54)} \\ &= -\frac{1}{m_i^n}\sum_\alpha \mathbf{v}_{i\alpha}^n \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^n}. && \text{(A.55)} \end{aligned}$$

A.4 Backward Grid to Particle (G2P)

(F, G2P) For $\mathbf{v}_p^n$, we have

$$\begin{aligned} \mathbf{p}_i^n &= \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)\left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2}\Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n\right)(\mathbf{x}_i - \mathbf{x}_p^n)\right] && \text{(A.56)} \\ \Longrightarrow \frac{\partial L}{\partial \mathbf{v}_{p\alpha}^n} &= \left[\sum_i \frac{\partial L}{\partial \mathbf{p}_p^n}\frac{\partial \mathbf{p}_p^n}{\partial \mathbf{v}_p^n}\right]_\alpha && \text{(A.57)} \\ &= \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) m_p \frac{\partial L}{\partial \mathbf{p}_{i\alpha}^n}. && \text{(A.58)} \end{aligned}$$

(G, G2P) For $\mathbf{P}_p^n$, we have

$$\begin{aligned}
\mathbf{p}_i^n &= \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)\left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2}\Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n\right)(\mathbf{x}_i - \mathbf{x}_p^n)\right] && \text{(A.59)}\\
\Longrightarrow \frac{\partial L}{\partial \mathbf{P}_{p\alpha\beta}^n} &= \left[\frac{\partial L}{\partial \mathbf{p}_i^n}\frac{\partial \mathbf{p}_i^n}{\partial \mathbf{P}_p^n}\right]_{\alpha\beta} && \text{(A.60)}\\
&= -\sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\frac{4}{\Delta x^2}\Delta t V_p^0 \sum_\gamma \frac{\partial L}{\partial \mathbf{p}_{i\alpha}^n}\mathbf{F}_{p_\gamma\beta}^n(\mathbf{x}_{i\gamma} - \mathbf{x}_{p\gamma}^n). && \text{(A.61)}
\end{aligned}$$

(H, G2P) For $\mathbf{F}_p^n$, we have

$$\begin{aligned}
\mathbf{F}_p^{n+1} &= (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1})\mathbf{F}_p^n && \text{(A.62)}\\
\mathbf{P}_p^n &= \mathbf{P}_p^n(\mathbf{F}_p^n) + \mathbf{F}_p \boldsymbol{\sigma}_{pa}^n && \text{(A.63)}\\
\mathbf{p}_i^n &= \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)\left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2}\Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n\right)(\mathbf{x}_i - \mathbf{x}_p^n)\right] && \text{(A.64)}\\
\Longrightarrow \frac{\partial L}{\partial \mathbf{F}_{p\alpha\beta}^n} &= \left[\frac{\partial L}{\partial \mathbf{F}_p^{n+1}}\frac{\partial \mathbf{F}_p^{n+1}}{\partial \mathbf{F}_p^n} + \frac{\partial L}{\partial \mathbf{P}_p^n}\frac{\partial \mathbf{P}_p^n}{\partial \mathbf{F}_p^n} + \frac{\partial L}{\partial \mathbf{p}_i^n}\frac{\partial \mathbf{p}_i^n}{\partial \mathbf{F}_p^n}\right]_{\alpha\beta} && \text{(A.65)}\\
&= \sum_\gamma \frac{\partial L}{\partial \mathbf{F}_{p\gamma\beta}^{n+1}}(\mathbf{I}_{\gamma\alpha} + \Delta t \mathbf{C}_{p\gamma\alpha}^{n+1}) + \sum_\gamma \sum_\eta \frac{\partial L}{\partial \mathbf{P}_{p\gamma\eta}}\frac{\partial^2 \Psi_p}{\partial \mathbf{F}_{p\gamma\eta}^n \partial \mathbf{F}_{p\alpha\beta}^n} && \text{(A.66)}\\
&+ \sum_\gamma \frac{\partial L}{\partial \mathbf{P}_{p\alpha\gamma}^n}\boldsymbol{\sigma}_{pa\beta\gamma} && \text{(A.67)}\\
&\quad + \sum_i -N(\mathbf{x}_i - \mathbf{x}_p^n)\sum_\gamma \frac{\partial L}{\partial \mathbf{p}_{i\gamma}^n}\frac{4}{\Delta x^2}\Delta t V_p^0 \mathbf{P}_{p\gamma\beta}^n(\mathbf{x}_{i\alpha} - \mathbf{x}_{p\alpha}^n). && \text{(A.68)}
\end{aligned}$$

(I, G2P) For $\mathbf{C}_p^n$, we have

$$\begin{aligned}
\mathbf{p}_i^n &= \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)\left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2}\Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n\right)(\mathbf{x}_i - \mathbf{x}_p^n)\right] && \text{(A.69)}\\
\Longrightarrow \frac{\partial L}{\partial \mathbf{C}_{p\alpha\beta}^n} &= \left[\sum_i \frac{\partial L}{\partial \mathbf{p}_i^n}\frac{\partial \mathbf{p}_i^n}{\partial \mathbf{C}_p^n}\right]_{\alpha\beta} && \text{(A.70)}\\
&= \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\frac{\partial L}{\partial \mathbf{p}_{i\alpha}^n} m_p(\mathbf{x}_{i\beta} - \mathbf{x}_{p\beta}^n). && \text{(A.71)}
\end{aligned}$$

(J, G2P) For $\mathbf{x}_p^n$, we have

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \tag{A.72}$$

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n \tag{A.73}$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2}\sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n(\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{A.74}$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)\left[m_p\mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2}\Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n\right)(\mathbf{x}_i - \mathbf{x}_p^n)\right] \tag{A.75}$$

$$m_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) m_p \tag{A.76}$$

$$\mathbf{G}_p := \left(-\frac{4}{\Delta x^2}V_p^0 \Delta t \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n\right) \tag{A.77}$$

$$\Longrightarrow \tag{A.78}$$

$$\frac{\partial L}{\partial \mathbf{x}_{p\alpha}^n} = \left[\frac{\partial L}{\partial \mathbf{x}_p^{n+1}}\frac{\partial \mathbf{x}_p^{n+1}}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial \mathbf{v}_p^{n+1}}\frac{\partial \mathbf{v}_p^{n+1}}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial \mathbf{C}_p^{n+1}}\frac{\partial \mathbf{C}_p^{n+1}}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial \mathbf{p}_i^n}\frac{\partial \mathbf{p}_i^n}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial m_i^n}\frac{\partial m_i^n}{\partial \mathbf{x}_p^n}\right]_\alpha \tag{A.79}$$

$$= \frac{\partial L}{\partial \mathbf{x}_{p\alpha}^{n+1}} \tag{A.80}$$

$$+ \sum_i \sum_\beta \frac{\partial L}{\partial \mathbf{v}_{p\beta}^{n+1}}\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}}\mathbf{v}_{i\beta}^n \tag{A.81}$$

$$+ \sum_i \sum_\beta \tag{A.82}$$

$$\frac{4}{\Delta x^2}\left\{-\frac{\partial L}{\partial \mathbf{C}_{p\beta\alpha}^{n+1}}N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_{i\beta} + \sum_\gamma \frac{\partial L}{\partial \mathbf{C}_{p\beta\gamma}^{n+1}}\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}}\mathbf{v}_{i\beta}(\mathbf{x}_{i\gamma} - \mathbf{x}_{p\gamma})\right\} \tag{A.83}$$

$$+ \sum_i \sum_\beta \tag{A.84}$$

$$\frac{\partial L}{\partial \mathbf{p}_{i\beta}^n}\left[\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}}\left(m_p\mathbf{v}_{p\beta}^n + [\mathbf{G}_p(\mathbf{x}_i - \mathbf{x}_p^n)]_\beta\right) - N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{G}_{p\beta\alpha}\right] \tag{A.85}$$

$$+ m_p \sum_i \frac{\partial L}{\partial m_i^n}\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}} \tag{A.86}$$

(K, G2P) For $\boldsymbol{\sigma}_{pa}^n$, we have

$$\mathbf{P}_p^n = \mathbf{P}_p^n(\mathbf{F}_p^n) + \mathbf{F}_p\boldsymbol{\sigma}_{pa}^n \tag{A.87}$$

$$\Longrightarrow \frac{\partial L}{\partial \boldsymbol{\sigma}_{pa\alpha\beta}^n} = \left[\frac{\partial L}{\partial \mathbf{P}_p^n}\frac{\partial \mathbf{P}_p^n}{\partial \boldsymbol{\sigma}_{p\alpha}^n}\right]_{\alpha\beta} \tag{A.88}$$

$$= \sum_\gamma \frac{\partial L}{\partial \mathbf{P}_{p\gamma\beta}^{n+1}}\mathbf{F}_{p\gamma\alpha}^n. \tag{A.89}$$

A.5 Friction Projection Gradients

When there are boundary conditions:

(L, grid) For $\mathbf{v}_i^n$, we have

$$
\begin{aligned}
l_{i\mathbf{n}} &= \sum_\alpha \mathbf{v}_{i\alpha}\mathbf{n}_{i\alpha} && \text{(A.90)}\\
\mathbf{v}_{i\mathbf{t}} &= \mathbf{v}_i - l_{i\mathbf{n}}\mathbf{n}_i && \text{(A.91)}\\
l_{i\mathbf{t}} &= \sqrt{\sum_\alpha \mathbf{v}_{i\mathbf{t}\alpha}^2 + \varepsilon} && \text{(A.92)}\\
\hat{\mathbf{v}}_{i\mathbf{t}} &= \frac{1}{l_{i\mathbf{t}}}\mathbf{v}_{i\mathbf{t}} && \text{(A.93)}\\
l_{i\mathbf{t}}^* &= \max\{l_{i\mathbf{t}} + c_i \min\{l_{i\mathbf{n}}, 0\}, 0\} && \text{(A.94)}\\
\mathbf{v}_i^* &= l_{i\mathbf{t}}^*\hat{\mathbf{v}}_{i\mathbf{t}} + \max\{l_{i\mathbf{n}}, 0\}\mathbf{n}_i && \text{(A.95)}\\
H(x) &:= [x \geq 0] && \text{(A.96)}\\
R &:= l_{i\mathbf{t}} + c_i \min\{l_{i\mathbf{n},0}\} && \text{(A.97)}\\
\Longrightarrow \frac{\partial L}{\partial l_{i\mathbf{t}}^*} &= \sum_\alpha \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^*}\hat{\mathbf{v}}_{i\mathbf{t}\alpha} && \text{(A.98)}\\
\frac{\partial L}{\partial \hat{\mathbf{v}}_{i\mathbf{t}}} &= \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^*} l_{i\mathbf{t}}^* && \text{(A.99)}\\
\frac{\partial L}{\partial l_{i\mathbf{t}}} &= -\frac{1}{l_{i\mathbf{t}}^2}\sum_\alpha \mathbf{v}_{i\mathbf{t}\alpha}\frac{\partial L}{\partial \hat{\mathbf{v}}_{i\mathbf{t}\alpha}} + \frac{\partial L}{\partial l_{i\mathbf{t}}^*}H(R) && \text{(A.100)}\\
\frac{\partial L}{\partial \mathbf{v}_{i\mathbf{t}\alpha}} &= \frac{\mathbf{v}_{i\mathbf{t}\alpha}}{l_{i\mathbf{t}}}\frac{\partial L}{\partial l_{i\mathbf{t}}} + \frac{1}{l_{i\mathbf{t}}}\frac{\partial L}{\partial \hat{\mathbf{v}}_{i\mathbf{t}\alpha}} && \text{(A.101)}\\
&= \frac{1}{l_{i\mathbf{t}}}\left[\frac{\partial L}{\partial l_{i\mathbf{t}}}\mathbf{v}_{i\mathbf{t}\alpha} + \frac{\partial L}{\partial \hat{\mathbf{v}}_{i\mathbf{t}\alpha}}\right] && \text{(A.102)}\\
\frac{\partial L}{\partial l_{i\mathbf{n}}} &= -\left[\sum_\alpha \frac{\partial L}{\partial \mathbf{v}_{i\mathbf{t}\alpha}}\mathbf{n}_{i\alpha}\right] && \text{(A.103)}\\
&\quad + \frac{\partial L}{\partial l_{i\mathbf{t}}^*}H(R)c_i H(-l_{i\mathbf{n}}) + \sum_\alpha H(l_{i\mathbf{n}})\mathbf{n}_{i\alpha}\frac{\partial L}{\partial \mathbf{v}_{i\alpha}^*} && \text{(A.104)}\\
\frac{\partial L}{\partial \mathbf{v}_{i\alpha}} &= \frac{\partial L}{\partial l_{i\mathbf{n}}}\mathbf{n}_{i\alpha} + \frac{\partial L}{\partial \mathbf{v}_{i\mathbf{t}\alpha}} && \text{(A.105)}
\end{aligned}
$$

Bibliography

[1] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *NIPS*, 2016.

[2] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *ICLR 2016*, 2016.

[3] Gilles Daviet and Florence Bertails-Descoubes. A semi-implicit material point method for the continuum simulation of granular materials. *ACM Transactions on Graphics (TOG)*, 35(4):102, 2016.

[4] Filipe de Avila Belbute-Peres, Kevin A Smith, Kelsey Allen, Joshua B Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. In *Neural Information Processing Systems*, 2018.

[5] Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *arXiv preprint arXiv:1611.01652*, 2016.

[6] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *ICRA*, pages 4397–4404. IEEE, 2015.

[7] Yu Fang, Yuanming Hu, Shi-min Hu, and Chenfanfu Jiang. A temporally adaptive material point method with regional time stepping. *Computer graphics forum*, 37, 2018.

[8] Marco Frigerio, Jonas Buchli, Darwin G. Caldwell, and Claudio Semini. RobCoGen: a code generator for efficient kinematics and dynamics of articulated robots, based on Domain Specific Languages. 7(1):36–54, 2016.

[9] Ming Gao, Andre Pradhana, Xuchen Han, Qi Guo, Grant Kot, Eftychios Sifakis, and Chenfanfu Jiang. Animating fluid sediment mixture in particle-laden flows. *ACM Trans. Graph.*, 37(4):149:1–149:11, July 2018.

[10] Ming Gao, Andre Pradhana Tampubolon, Chenfanfu Jiang, and Eftychios Sifakis. An adaptive generalized interpolation material point method for simulating elastoplastic materials. *ACM Trans. Graph.*, 36(6):223:1–223:12, November 2017.

[11] Ming Gao, Wang, Kui Wu, Andre Pradhana-Tampubolon, Eftychios Sifakis, Yuksel Cem, and Chenfanfu Jiang. Gpu optimization of material point methods. *ACM Transactions on Graphics (TOG)*, 32(4):102, 2013.

[12] J Gaume, T Gast, J Teran, A van Herwijnen, and C Jiang. Dynamic anticrack propagation in snow. *Nature communications*, 9(1):3047, 2018.

[13] Qi Guo, Xuchen Han, Chuyuan Fu, Theodore Gast, Rasmus Tamstorf, and Joseph Teran. A material point method for thin shells with frictional contact. *ACM Transactions on Graphics (TOG)*, 37(4):147, 2018.

[14] Susumu Hara, Tetsuji Zama, Wataru Takashima, and Keiichi Kaneto. Artificial muscles based on polypyrrole actuators with large strain and stress induced electrically. *Polymer journal*, 36(2):151, 2004.

[15] Yuanming Hu. Taichi: An open-source computer graphics library. *arXiv preprint arXiv:1804.09293*, 2018.

[16] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics*, 37(4):150, 2018.

[17] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin. The affine particle-in-cell method. *ACM Trans Graph*, 34(4):51:1–51:10, 2015.

[18] Chenfanfu Jiang, Theodore Gast, and Joseph Teran. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Transactions on Graphics (TOG)*, 36(4):152, 2017.

[19] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, page 24. ACM, 2016.

[20] Steven G Johnson. The nlopt nonlinear-optimization package, 2014.

[21] Gergely Klár, Theodore Gast, Andre Pradhana, Chuyuan Fu, Craig Schroeder, Chenfanfu Jiang, and Joseph Teran. Drucker-prager elastoplasticity for sand animation. *ACM Transactions on Graphics (TOG)*, 35(4):103, 2016.

[22] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4):104, 2013.

[23] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4):153, 2014.

[24] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. *arXiv:1806.08047*, 2018.

[25] Daniel Ram, Theodore Gast, Chenfanfu Jiang, Craig Schroeder, Alexey Stomakhin, Joseph Teran, and Pirouz Kavehpour. A material point method for viscoelastic fluids, foams and sponges. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 157–163. ACM, 2015.

[26] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *ICML*, 2018.

[27] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. *arXiv:1806.06094*, 2018.

[28] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)*, 32(4):102, 2013.

[29] Deborah Sulsky, Shi-Jian Zhou, and Howard L Schreyer. Application of a particle-in-cell method to solid mechanics. *Computer physics communications*, 87(1-2):236–252, 1995.

[30] Andre Pradhana Tampubolon, Theodore Gast, Gergely Klár, Chuyuan Fu, Joseph Teran, Chenfanfu Jiang, and Ken Museth. Multi-species simulation of porous sand and water mixtures. *ACM Trans. Graph.*, 36(4):105:1–105:11, July 2017.

[31] Russ Tedrake and the Drake Development Team. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2016.

[32] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[33] Marc Toussaint, Kelsey R Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *RSS*, 2018.