ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics

Yuanming Hu MIT CSAIL
Jiancheng Liu * Tsinghua University, IIIS
Andrew Spielberg * MIT CSAIL
Joshua B. Tenenbaum MIT CSAIL
William T. Freeman MIT CSAIL
Jiajun Wu MIT CSAIL
Daniela Rus MIT CSAIL
Wojciech Matusik MIT CSAIL

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 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 tasks for soft robotics.

1 Introduction

Robot planning and control algorithms often rely on physical simulators for prediction and optimization [Erez et al., 2015]. *Differentiable* physical simulators enable the use of gradient-based optimization methods, significantly improving the efficiency and precision by which controllers can be learned. Motivated by this, there has been extensive research on differentiable rigid body simulators, using approximate [Chang et al., 2016] and exact [Degrave et al., 2016, Frigerio et al., 2016] methods.

We introduce a real-time, differentiable physical simulator for deformable objects, building upon the Moving Least Squares Material Point Method (MLS-MPM) [Hu et al., 2018]. 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 [Sulsky et al., 1995]. 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 parameters, with both forward simulation and back-propagation running efficiently on GPUs.

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, design, and system

*equally contributed. Project page: https://github.com/yuanming-hu/ChainQueen
[†]Or 乾坤 , literally "everything between the sky and the earth."

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

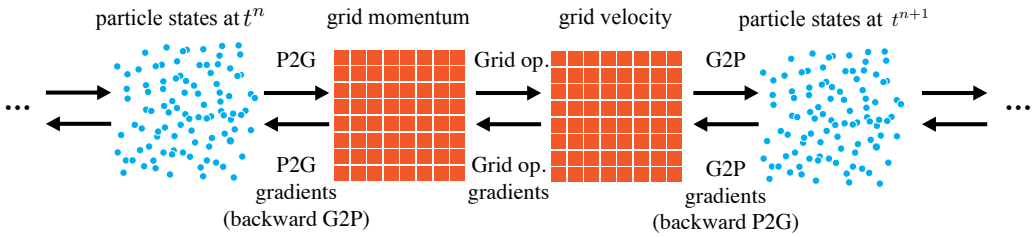

Figure 1: One time step of MLS-MPM. Top arrows are for forward simulation and bottom ones are for back propagation. A controller is embedded to generate actuation given particle configurations.

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.

2 Forward simulation and back-propagation

We use the moving least squares material point method (MLS-MPM) [Hu et al., 2018] to discretize continuum mechanics, which is governed by momentum conservation $\left(\frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \mathbf{g} \right)$ and mass conservation $\left(\frac{D}{Dt} + \nabla \cdot \mathbf{v} = 0 \right)$. We briefly cover the basics of MLS-MPM and readers are referred to Jiang et al. [2016] and Hu et al. [2018] 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. The MLS-MPM simulation cycle has three steps:

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) [Jiang et al., 2015] and moving least squares force discretization [Hu et al., 2018].

2. **Grid operations.** Grid momentum is normalized into grid velocity after division by grid mass. Note that neighbouring particles interact with each other through their shared grid nodes, and collisions are handled automatically.

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 $\mathbf{x}_p^{n+1}$. The constitutive model properties are updated.

Based on the gradients we have derived analytically, we have designed a high-performance implementation.Gradients of particle states 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, iteratively applying the chain rule one level higher allows for backward propagation yields gradients of the final state of a full simulation with respect to its initial state. That application of the chain rule simultaneously yields gradients with respect to the controller parameters that are used in each state. Similar application of the chain rule yields gradients with respect to design parameters. We have designed a simple high-level TensorFlow interface on which end-users can build their applications.

Our high-performance implementation[‡] takes advantage of the computational power of modern GPUs through CUDA. Our CUDA simulator is $4-9\times$ faster than Flex and $132\times$ faster than the TensorFlow reference version. We designed five test cases to evaluate the accuracy of both forward simulation and backward gradient evaluation. The relative error is within 3×10^{-5} even for simulations with as many as 1000 time steps.

[‡]Based the **Taichi** [Hu, 2018] open source computer graphics library.

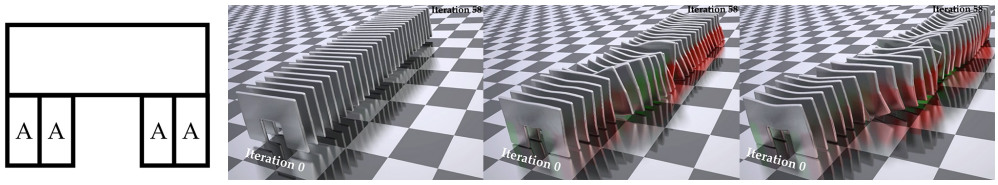

Figure 2: A soft 2D walker with controller optimized using gradient descent, aiming to achieve a maximum distance. It has four actuators (left, marked by letter 'A's) with each capable of stretching or compressing in the vertical direction. We visualize the stacked trajectories throughout the optimization process.

3 Gradient-based Policy Optimization

The most attractive feature of our simulator is the existence of quickly computable gradients, which allows the use of efficient gradient-based optimization algorithms. In this section, we show the effectiveness of our differentiable simulator on policy and trajectory optimization tasks.

We can optimize regression-based controller models for soft robots and efficiently discover stable gaits. The controller takes as input the state vector z, which includes target position, the center of mass position, and velocity of each composed soft component. In our examples, the actuation vector a for up to 16 actuators is generated by the controller. During optimization, we perform gradient descent on variables W and b, where $\mathsf{a} = \tanh(\mathsf{W}\mathsf{z} + \mathsf{b})$ is the actuation-generating controller.

Similarly, we can optimize open-loop trajectories, useful for particularly nonlinear problems. Here, the simulation is broken up into N time windows, and an explicit actuation vector a_N is solved for each. Consraints on starting and ending actuation, Similarly, design variables , including, for instance, the Young's modulus of each particle can be solved for in the same optimization.

We have designed a series of experiments (Fig. 2 and Fig. 3a). Gradient-based optimizers successfully compute desired closed loop controllers controllers within only tens or hundreds of iterations. Similarly, open-loop controllers and designs can be solved in a few tens of major iterations of sequential-quadratic programming algorithms, as in Fig. 3b.

Finally, we present a few simple system identification examples demonstrating the potential of our simulator for inference tasks; see https://youtu.be/4IWD4iGIsB4 for more details.

To emphasize the merits of gradient-based approaches, we compare our control method with proximal policy optimization (PPO) [Schulman et al., 2017], a state-of-the-art reinforcement learning algorithm. PPO is an actor-critic method which relies on sampled gradients of a *reward* function in order to optimize a policy. This sampling-based approach is model-free; it relies on gradients of the rewards with respect to controller parameters, but *not* with respect to the physical model for updates. For our comparison, we use velocity projected onto the direction toward the goal as the reward. As velocity toward the goal increases, final distance to the goal decreases. We use a simplified single link version (with only two adjacent actuators) runner as a benchmark. Quantitative results for the finger are shown in Fig. 4, demonstrating that for certain soft locomotion tasks our gradient-based method can be more efficient than model-free approaches.

References

Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *ICLR 2016*, 2016. 1

Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *arXiv preprint arXiv:1611.01652*, 2016. 1

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. 1

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. 1