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

Yuanming Hu, Jiancheng Liu*, Andrew Spielberg*,
Joshua B. Tenenbaum, William T. Freeman, Jiajun Wu, Daniela Rus, Wojciech Matusik[1,2]

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. Therefore, rigid body simulators and recently their differentiable variants are studied extensively. 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 with collisions and can be seamlessly incorporated into soft robotic 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 inference, control and co-design tasks for soft robotics.**

I. INTRODUCTION

Robot planning and control algorithms often rely on physical simulators for prediction and optimization [1], [2]. 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 [3], [4] and exact [5], [6], [7] 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 paper, we introduce a real-time, differentiable physical simulator for deformable objects, using the Moving Least Squares Material Point Method (MLS-MPM) [8] for

forward simulation. We additionally implement analytically derived gradients of MLS-MPM to make it differentiable. 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 [9]. 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. 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 soft object simulator. Numerical and experimental validation suggest that ChainQueen achieves high precision in both forward simulation and backward gradient computation.

ChainQueen's support for differentiation (i.e. gradient computation) is what makes it different from existing soft object simulators (e.g., the recent work on MLS-MPM [8]). Such capability 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.

[1]Y. Hu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik are with Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA.

[2]J. Liu is with Institute for Interdisciplinary Information Science, Tsinghua University, Beijing, China.

*J. Liu and A. Spielberg are joint second authors.

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

II. Related Work

A. Material Point Method

The material point method has been extensively developed from both a solid mechanics [9] and computer graphics [10] perspective. As a hybrid Eulerian-Langrangian method, MPM has demonstrated its versatility in simulating snow [11], [12], sand [13], [14], non-Newtonion fluids [15], cloth [16], [17], solid-fluid coupling [18], [19], rigid body coupling, and cutting [8]. [20] 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 [21] by Gao et al., from which we borrow many useful optimization practices. Though efficient when solving forward simulation, their simulator does not support gradient evaluation, 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.

B. Differentiable Simulation and Control

Recently, there has been an increasing interest in building differentiable simulators for planning and control. For rigid bodies, [22], [3] and [4] proposed to approximate object interaction with neural nets; later, [23] explored their usage in control. Approximate analytic differentiable rigid body simulators have also been proposed [5], [24]. Such systems have been deployed for manipulation and planning [25].

Differentiable simulators for deformable objects have been less studied. Recently, [26] proposed SPNets for differentiable simulation of position-based fluids [27]. 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 [4]. 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.

III. Forward simulation and back-propagation

We use the moving least squares material point method (MLS-MPM) [8] to discretize continuum mechanics, which

TABLE I: 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 [28]
$\mathbf{P}_p$	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

is governed by the following two equations:

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g} \quad \text{(momentum conservation)}, \quad (1)$$

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v} = 0 \quad \text{(mass conservation)}. \quad (2)$$

We briefly cover the basics of MLS-MPM and readers are referred to [10] and [8] 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 I. 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:

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) [28] and moving least squares force discretization [8], weighted by a compact B-spline kernel N: ($\mathbf{G}_p^n$ below is a temporary tensor)

$$m_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) m_p, \quad (3)$$

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

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n + \mathbf{G}_p^n (\mathbf{x}_i - \mathbf{x}_p^n) \right]. (5)$$

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

$$\mathbf{v}_i^n = \frac{1}{m_i^n} \mathbf{p}_i^n. \quad (6)$$

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

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
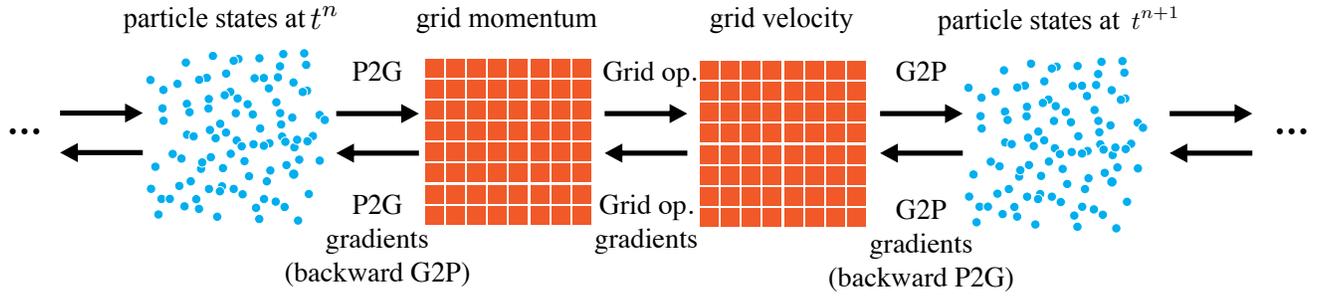
6266

Fig. 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.

(e.g. deformation gradients $\mathbf{F}_p^{n+1}$) are updated.

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n, \tag{7}$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2}\sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n(\mathbf{x}_i - \mathbf{x}_p^n)^T, \tag{8}$$

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t\mathbf{C}_p^{n+1})\mathbf{F}_p^n, \tag{9}$$

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t\mathbf{v}_p^{n+1}. \tag{10}$$

For soft robotics, we additionally introduce an actuation model. Inspired by actuators such as [29], 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 $\boldsymbol{\sigma}_{pa} = \mathrm{Diag}(a_x, a_y, a_z)$ – the stress in the material space. This framework supports the use of other differentiable actuation models including pneumatic, hydraulic, and cable-driven actuators. Fig. 1 illustrates forward simulation and back propagation.

Theoretically, MLS-MPM is naturally differentiable. Though the forward direction has been extensively used in computer graphics, the backward direction (differentiation or back-propagation) is largely unexplored in practice.

Based on the gradients we 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. 2).

Our high-performance implementation[‡] takes advantage of the computational power of modern GPUs through CUDA. We also implemented a reference implementation in TensorFlow.

[†]Please see the supplemental document (https://github.com/yuanming-hu/ChainQueen) for the gradient derivations.

[‡]Based the **Taichi** [30] open source computer graphics library.

TABLE II: 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. "Flex" refers to [31], the real-time simulator from NVIDIA.

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**)

Note that programming physical simulation as a "computation graph" using high-level frameworks such as TensorFlow is less efficient. In fact, when all the overheads are gone, our optimized CUDA solver is $132\times$ 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. In contrast, our CUDA implementation is tailored for MLS-MPM and explicitly optimized for parallelism and locality, thus delivering high-performance.

IV. EVALUATION

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

A. 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 [31], a popular Position-based Dynamics (PBD) physical simulator capable of
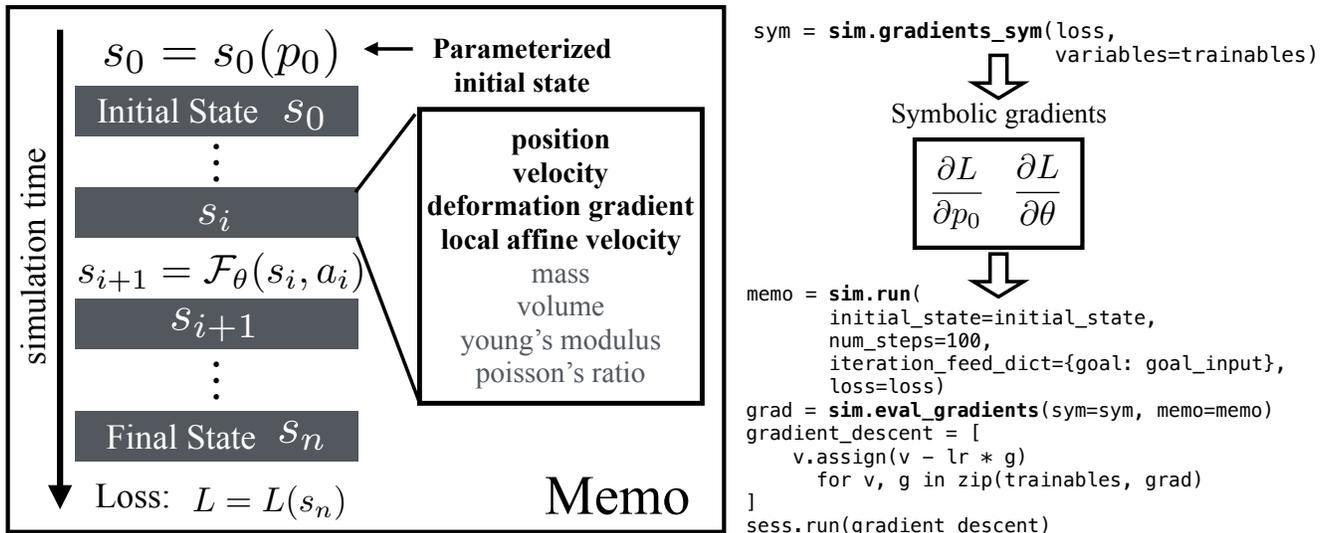
6267

Fig. 2: 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).

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 ratio 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 II. 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.

B. Accuracy

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

TABLE III: Relative error in forward simulation and gradient precision. Empty values are because of too short time for collision to happen. Analytic ground truth can be computed for A1 and A2. We use double precision for accurate finite difference results as the ground truth for case B and C.

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}

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 15 cm height, and compared its dynamic motion to simulation.

Results for A1, A2, B and C are listed in Table III. 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. For the experimental comparisons, the results are shown in Fig. 3.

V. INFERENCE, CONTROL AND CO-DESIGN

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.

A. Physical Parameter Inference

ChainQueen can be used to infer physical system properties given its observed motion, e.g. perform gradient descent to
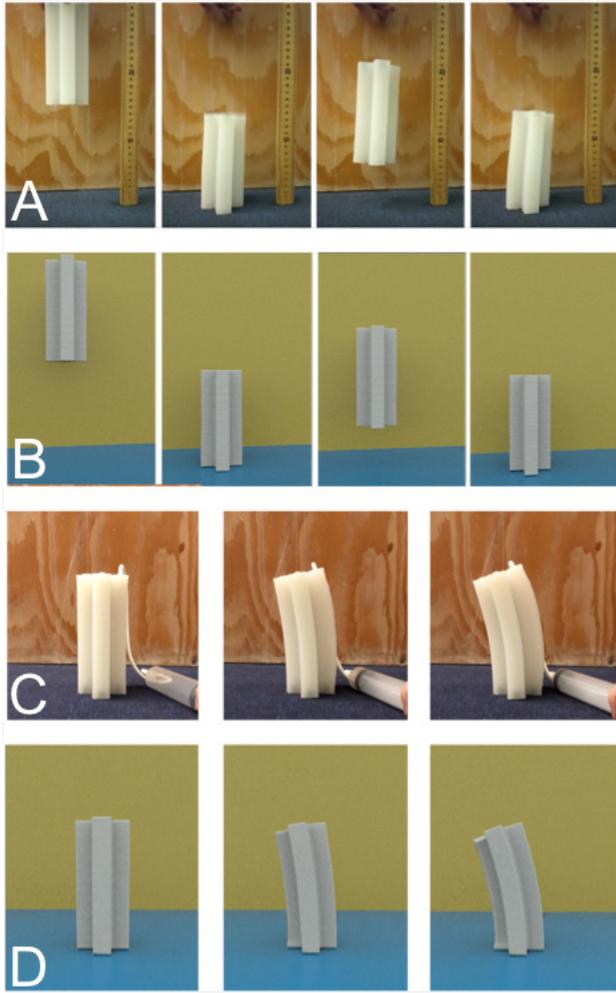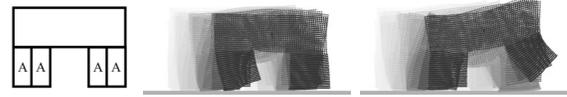
Fig. 4: A soft 2D walker with controller optimized using gradient descent, aiming to achieve a maximum distance after 600 simulation steps. The walker has four actuators (left, marked by letter 'A's) with each capable of stretching or compressing in the vertical direction. The full walking animation (middle and right) is available in the video.

Fig. 3: Experiments on the pneumatic leg. **Row (A, B)** Experiment and simulation of a bouncing experiment with the leg dropping at 15 cm. **Row (C, D)** Experiment and simulation of the actuation test.

infer the relative densities of two colliding elastic balls (see figure above, 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.

B. 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. 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 including the 2D biped runner (Fig. 4) and robotic finger, and 3D quadrupedal

runner (Fig. 6), crawler and robotic arm. Gradient-based optimizers successfully compute desired controllers within only *tens or hundreds* of iterations. Visual results are included in the supplemental video.

To emphasize the merits of gradient-based approaches, we compare our control method with proximal policy optimization (PPO) [32], 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. [§] We use a simplified single link version (with only two adjacent actuators) of Fig. 5 and the 2D runner Fig. 4 as a benchmark. Quantitative results for the finger are shown in Fig. 7. We performed a similar comparison on the 2D walker, the controller optimized by ChainQueen for the 2D walker starts functioning well within 20 minutes; by comparison the policy recovered by PPO still chose nearly-random actions after over 4 hours of training; demonstrating that for certain soft locomotion tasks our gradient-based method can be more efficient than model-free approaches.

C. 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 [33]. We compared our co-design solution to

---

[§]Note that this is functionally extremely similar to a distance loss; the cumulative reward $\int_t = 0^T v_{goal} dt = D - \|x_T - x_{goal}\|$, where D is the initial distance and x_T and x_{goal} represent world coordinates of the robot at time T and of the goal, respectively. As velocity toward the goal increases, final distance to the goal decreases.

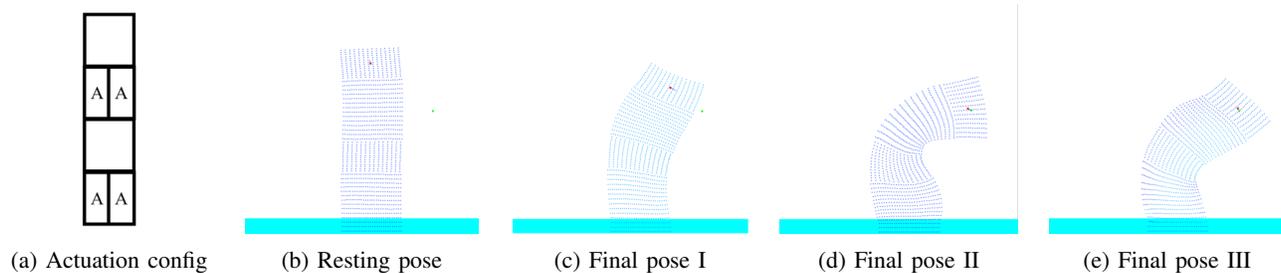| (a) Actuation config | (b) Resting pose | (c) Final pose I | (d) Final pose II | (e) Final pose III |

Fig. 5: **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 100% 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.

Fig. 6: A 3D quadrupedal runner. Please see the supplemental video for more visual results.
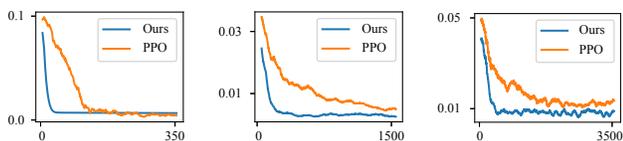

Fig. 7: Gradient-free optimization using PPO and gradient-descent based on ChainQueen, on the 2D finger task. Thanks to the accurate gradient information, even the most vanilla optimizer can beat state-of-the-art reinforcement learning algorithms by one order of magnitude regarding optimization speed. (Left) single, fixed target. (Middle) random targets. (Right) random targets, larger range. Curves are smoothed over 10, 100 and 100 iterations respectively. The x-axis is simulation rounds (forward only for PPO, forward+backward for ours) and y-axis the loss.

fixed designs. The designed stiffness distribution is shown in Fig. 5, along with controls. The convergence for the different tasks can be seen in Fig. 8. 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 ranges 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.

VI. 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 plan to open source.
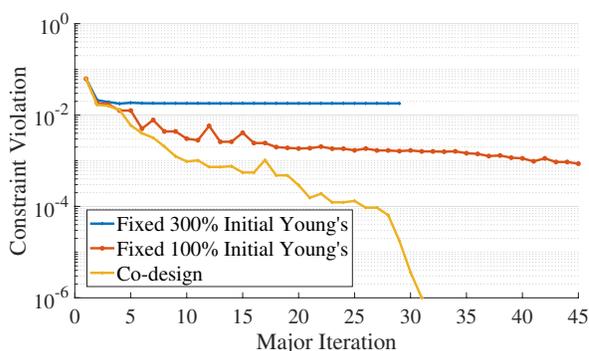

Fig. 8: Convergence of the arm reaching task for co-design vs. fixed arm designs. The fixed designs can make progress but not complete the task, while with co-design, the task can be completed and the actuation cost is lower. Constraint violation is the norm of two constraints: distance of end-effector to goal and mean squared velocity of the particles.

One interesting future direction is to couple our soft object simulation with rigid body simulation, as done in [8]. As derived in [34], 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 [8] can be an interesting direction.

REFERENCES

[1] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*. IEEE, 2012, pp. 5026–5033.

[2] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *ICRA*. IEEE, 2015, pp. 4397–4404.

[3] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional object-based approach to learning physical dynamics," *ICLR 2016*, 2016.

[4] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, and D. L. Yamins, "Flexible neural representation for physics prediction," *arXiv:1806.08047*, 2018.

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

[6] R. Tedrake and the Drake Development Team, "Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems," 2016. [Online]. Available: https://drake.mit.edu

[7] M. Frigerio, J. Buchli, D. G. Caldwell, and C. Semini, "RobCoGen: a code generator for efficient kinematics and dynamics of articulated robots, based on Domain Specific Languages," vol. 7, no. 1, pp. 36–54, 2016.

[8] Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang, "A moving least squares material point method with displacement discontinuity and two-way rigid body coupling," *ACM TOG*, vol. 37, no. 4, p. 150, 2018.

[9] D. Sulsky, S.-J. Zhou, and H. L. Schreyer, "Application of a particle-in-cell method to solid mechanics," *Computer physics communications*, vol. 87, no. 1-2, pp. 236–252, 1995.

[10] C. Jiang, C. Schroeder, J. Teran, A. Stomakhin, and A. Selle, "The material point method for simulating continuum materials," in *ACM SIGGRAPH 2016 Courses*. ACM, 2016, p. 24.

[11] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle, "A material point method for snow simulation," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 102, 2013.

[12] J. Gaume, T. Gast, J. Teran, A. van Herwijnen, and C. Jiang, "Dynamic anticrack propagation in snow," *Nature communications*, vol. 9, no. 1, p. 3047, 2018.

[13] G. Klár, T. Gast, A. Pradhana, C. Fu, C. Schroeder, C. Jiang, and J. Teran, "Drucker-prager elastoplasticity for sand animation," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 103, 2016.

[14] G. Daviet and F. Bertails-Descoubes, "A semi-implicit material point method for the continuum simulation of granular materials," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 102, 2016.

[15] D. Ram, T. Gast, C. Jiang, C. Schroeder, A. Stomakhin, J. Teran, and P. Kavehpour, "A material point method for viscoelastic fluids, foams and sponges," in *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2015, pp. 157–163.

[16] C. Jiang, T. Gast, and J. Teran, "Anisotropic elastoplasticity for cloth, knit and hair frictional contact," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 152, 2017.

[17] Q. Guo, X. Han, C. Fu, T. Gast, R. Tamstorf, and J. Teran, "A material point method for thin shells with frictional contact," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 147, 2018.

[18] M. Gao, A. Pradhana, X. Han, Q. Guo, G. Kot, E. Sifakis, and C. Jiang, "Animating fluid sediment mixture in particle-laden flows," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 149:1–149:11, Jul. 2018.

[19] A. P. Tampubolon, T. Gast, G. Klár, C. Fu, J. Teran, C. Jiang, and K. Museth, "Multi-species simulation of porous sand and water mixtures," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 105:1–105:11, Jul. 2017.

[20] M. Gao, A. P. Tampubolon, C. Jiang, and E. Sifakis, "An adaptive generalized interpolation material point method for simulating elastoplastic materials," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 223:1–223:12, Nov. 2017.

[21] M. Gao, Wang, K. Wu, A. Pradhana-Tampubolon, E. Sifakis, Y. Cem, and C. Jiang, "Gpu optimization of material point methods," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 102, 2013.

[22] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *NIPS*, 2016.

[23] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *ICML*, 2018.

[24] F. de Avila Belbute-Peres, K. A. Smith, K. Allen, J. B. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Neural Information Processing Systems*, 2018.

[25] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *RSS*, 2018.

[26] C. Schenck and D. Fox, "Spnets: Differentiable fluid dynamics for deep neural networks," *arXiv:1806.06094*, 2018.

[27] M. Macklin and M. Müller, "Position based fluids," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 104, 2013.

[28] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin, "The affine particle-in-cell method," *ACM Trans Graph*, vol. 34, no. 4, pp. 51:1–51:10, 2015.

[29] S. Hara, T. Zama, W. Takashima, and K. Kaneto, "Artificial muscles based on polypyrrole actuators with large strain and stress induced electrically," *Polymer journal*, vol. 36, no. 2, p. 151, 2004.

[30] Y. Hu, "Taichi: An open-source computer graphics library," *arXiv preprint arXiv:1804.09293*, 2018.

[31] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 153, 2014.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[33] S. G. Johnson, "The nlopt nonlinear-optimization package," 2014.

[34] Y. Fang, Y. Hu, S.-m. Hu, and C. Jiang, "A temporally adaptive material point method with regional time stepping," *Computer graphics forum*, vol. 37, 2018.