PLASTICINELAB: A SOFT-BODY MANIPULATION BENCHMARK WITH DIFFERENTIABLE PHYSICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Simulated virtual environments serve as one of the main driving forces behind developing and evaluating skill learning algorithms. However, existing environments typically only simulate rigid body physics. Additionally, the simulation process usually does not provide gradients that might be useful for planning and control optimizations. We introduce a new differentiable physics benchmark called *PasticineLab*, which includes a diverse collection of soft body manipulation tasks. In each task, the agent uses manipulators to deform the plasticine into a desired configuration. The underlying physics engine supports *differentiable elastic and plastic deformation* using the DiffTaichi system, posing many under-explored challenges to robotic agents. We evaluate several existing reinforcement learning (RL) methods and gradient-based methods on this benchmark. Experimental results suggest that 1) RL-based approaches struggle to solve most of the tasks efficiently; 2) gradient-based approaches, by optimizing open-loop control sequences with the built-in differentiable physics engine, can rapidly find a solution within tens of iterations, but still fall short on multi-stage tasks that require long-term planning. We expect that PlasticineLab will encourage the development of novel algorithms that combine differentiable physics and RL for more complex physics-based skill learning tasks. PlasticineLab will be made publicly available [1].

1 INTRODUCTION

Virtual environments, such as Arcade Learning Environment (ALE) (Bellemare et al., 2013), MuJoCo (Todorov et al., 2012), and OpenAI Gym (Brockman et al., 2016), have significantly benefited the development and evaluation of learning algorithms on intelligent agent control and planning. However, existing virtual environments for skill learning typically involves *rigid-body* dynamics only. Research on establishing standard *soft-body* environments and benchmarks is sparse, despite the wide range of applications of soft bodies in multiple research fields, e.g., simulating virtual surgery in healthcare, modeling humanoid characters in computer graphics, developing biomimetic actuators in robotics, and analyzing fracture and tearing in material science.

Compared to its rigid-body counterpart, soft-body dynamics is much more intricate to simulate, control, and analyze. One of the biggest challenges comes from its infinite degrees of freedom (DoFs) and the corresponding high-dimensional governing equations. The intrinsic complexity of soft-body dynamics invalidates the direct application of many successful robotics algorithms designed for rigid bodies only and inhibits the development of a simulation benchmark for evaluating novel algorithms tackling soft-body tasks.

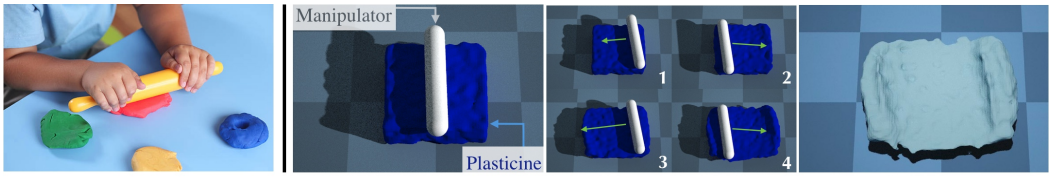
In this work, we aim to address this problem by proposing PlasticineLab, a novel benchmark for running and evaluating 10 soft-body manipulation tasks with 50 configurations in total. These tasks have to be performed by complex operations, including pinching, rolling, chopping, molding, and carving. Our benchmark is highlighted by the adoption of *differentiable physics* in the simulation environment, providing for the first time analytical gradient information in a soft-body benchmark, making it possible to conduct supervised learning with gradient-based optimization. In terms of the soft-body model, we choose to study plasticine (Fig. 1, left), a versatile elastoplastic material for sculpturing. Plasticine deforms elastically under small deformation, and plastically under large deformation. Compared to regular elastic soft bodies, plasticine establishes more diverse and realistic

[1] Project page: https://sites.google.com/view/plasticinelab

behaviors and brings challenges unexplored in previous research, making it a representative medium to test soft-body manipulation algorithms (Fig. 1, right).

We implement PlasticineLab, its gradient support, and its elastoplastic material model using Taichi (Hu et al., 2019a), whose CUDA backend leverages massive parallelism on GPUs to simulate a diverse collection of 3D soft-bodies in real time. We model the elastoplastic material using the Moving Least Squares Material Point Method (Hu et al., 2018) and the von Mises yield criterion. We use Taichi's two-scale reverse-mode differentiation system (Hu et al., 2020) to automatically compute gradients, including the numerically challenging SVD gradients brought by the plastic material model. With full gradients at hand, we evaluated gradient-based planning algorithms on all soft-robot manipulation tasks in PlasticineLab and compared its efficiency to RL-based methods. Our experiments revealed that gradient-based planning algorithms could find a more precious solution within tens of iterations with the extra knowledge of the physical model. At the same time, RL methods may fail even after 10K episodes. However, gradient-based methods lack enough momentum to resolve long-term planning, especially on multi-stage tasks. These findings have deepened our understanding of RL and gradient-based planning algorithms. Additionally, it suggests a promising direction of combining both families of methods' benefits to advance complex planning tasks involving soft-body dynamics. In summary, we contribute in this work the following:

- We introduce, to the best of our knowledge, the first skill learning benchmark involving elastic and plastic soft bodies.
- We develop a fully-featured differentiable physical engine, which supports elastic and plastic deformation, soft-rigid material interaction, and a tailored contact model for differentiability.
- The broad task coverage in the benchmark enables a systematic evaluation and analysis of representative RL and gradient-based planning algorithms. We hope such a benchmark can inspire future research to combine differentiable physics with imitation learning and RL.



Human | Initial State | Manipulation | Target Plasticine Shape

Figure 1: **Left:** A child deforming a piece of plasticine into a thin pie using a rolling pin. **Right:** The challenging **RollingPin** scene in PlasticineLab. The agent needs to flatten the material by rolling the pin back and forth, so that the plasticine deforms into the target shape.

2 Related Work

Learning in virtual environments Recently, several simulation platforms and datasets have been developed to facilitate the research and development of new algorithms on RL and robotics. An incomplete list includes RL Benchmark (Duan et al., 2016), DeepMind Lab (Beattie et al., 2016), OpenAI Gym (Brockman et al., 2016), AI2-THOR (Kolve et al., 2017), VirtualHome (Puig et al., 2018), Gibson (Xia et al., 2018), Habitat (Savva et al., 2019), SAPIEN (Xiang et al., 2020), and TDW (Gan et al., 2020). We observe a tendency to use full-physics simulators with realistic dynamics. However, most of these virtual environments are based on rigid-body physical engines, such as MuJoCo (Todorov et al., 2012) and PyBullet (Coumans & Bai, 2016). While some support soft-body dynamics in theory (e.g., TDW and SAPIEN is based on NVIDIA PhysX (PhysX) that supports particle simulation), none has provided the assets and tasks for soft-body manipulation. Differentiable information is also missing in these engines. We fill in this gap with our PlasticineLab benchmark.

Differentiable physics engines Differentiable physics engines for machine learning have gained increasing popularity. One family of approaches *approximates* physical simulators using neural networks, which are naturally differentiable (Battaglia et al., 2016; Chang et al., 2016; Mrowca et al., 2018; Li et al., 2018). A more direct and accurate approach is to implement physics-based simulators using differentiable programming systems, e.g., standard deep learning frameworks equipped with automatic differentiation tools (Degrave et al., 2016; de Avila Belbute-Peres et al., 2018; Schenck &

Fox, 2018; Heiden et al., 2019). These systems are typically redistricted to explicit time integration. Other approaches of evaluating simulation gradient computation include using the adjoint methods to differentiate implicit time integrators (Bern et al., 2019; Geilinger et al., 2020), LCP (de Avila Belbute-Peres et al., 2018) and leveraging QR decompositions (Liang et al., 2019; Qiao et al., 2020). Closely related to our work is ChainQueen (Hu et al., 2019b), a differentiable simulator for *elastic* bodies, and DiffTaichi (Hu et al., 2020), a system to automatically generate high-performance simulation gradient kernels. Our simulator is originated from ChainQueen but with significant modifications in order to add our novel support for *plasticity and contact* gradients.

Trajectory optimization Our usage of differentiable simulation in planning soft-body manipulation is closely related to trajectory optimization, a topic that has been extensively studied in robotics for years and has been applied to terrestrial robots (Posa et al., 2014; Erez & Todorov, 2012; de Avila Belbute-Peres et al., 2018), aerial robots (Foehn et al., 2017; Tang & Kumar, 2015; Sreenath et al., 2013), and, closest to examples in our work, robotic manipulators (Marchese et al., 2016; Li et al., 2015). Both trajectory optimization and differentiable physics formulate planning as an optimization problem and derive gradients from governing equations of the dynamics (Tedrake, 2020). Still, the problem of motion planning for soft-body manipulation is under exploration in both communities because of two challenges: first, the high degrees of freedom (DoFs) in soft-body dynamics make traditional trajectory optimization methods computationally prohibitive. Second, and more importantly, contacts between soft bodies are intricate to formulate in a concise manner. Our differentiable physics simulator addresses both issues with the recent development of DiffTaichi (Hu et al., 2020), unlocking gradient-based optimization techniques on planning for soft-body manipulation with high DoFs ($> 10,000$) and complex contact.

Learning-based soft body manipulation Finally, our work is also relevant to prior methods that propose learning-based techniques for manipulating physics systems with high degrees of freedom, e.g. cloth (Liang et al., 2019), fluids (Ma et al., 2018; Holl et al., 2020), and rope (Yan et al., 2020). Compared to our work, all of these prior papers focused on providing solutions to specific robot instances, while the goal of our work is to propose a comprehensive benchmark for evaluating and developing novel algorithms in soft-body research. There are also considerable works on soft manipulators (George Thuruthel et al., 2018; Della Santina et al., 2018). Different from them, we study soft body manipulation with rigid manipulators.

3 THE PLASTICINELAB LEARNING ENVIRONMENT

PlasticineLab is a collection of challenging soft-body manipulation tasks powered by a differentiable physics simulator. All tasks in PlasticineLab require an agent to deform one or more pieces of 3D plasticine with rigid-body manipulators. The underlying simulator in PlasticineLab allows users to execute complex operations on soft bodies, including pinching, rolling, chopping, molding, and carving. We introduce the high-level design of the learning environment in this section and leave the technical details of the underlying differentiable simulator in Sec. 4.

3.1 TASK REPRESENTATION

PlasticineLab presents 10 tasks with the focus on soft-body manipulation. Each task contains one or more soft bodies and a kinematic manipulator, and the end goal is to deform the soft body into a target shape with the planned motion of the manipulator. Following the standard reinforcement learning framework (Brockman et al., 2016), the agent is modeled with the Markov Decision Process (MDP), and the design of each task is defined by its state and observation, its action representation, its goal definition, and its reward function.

Markov Decision Process An MDP contains a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, and a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. In PlasticineLab, the physics simulator determines the transition between states. The goal of the agent is to find a stochastic policy $\pi(a|s)$ to sample action $a \in \mathcal{A}$ given state $s \in \mathcal{S}$, that maximizes the expected cumulative future return $E_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)\right]$ where $0 < \gamma < 1$ is the discount factor.

State The state of a task includes a proper representation of soft bodies and the end effector of the kinematic manipulator. Following the widely used particle-based simulation methodology in previous work, we represent soft-body objects as a particle system whose state includes its particles' positions, velocities, and strain and stress information. Specifically, the particle state is encoded as

a matrix of size $N_p \times d_p$ where N_p is the number of particles. Each row in the matrix consists of information from a single particle: two 3D vectors for position and velocities and two 3D matrices for deformation gradients and affine velocity fields (Jiang et al., 2015), all of which are stacked together and flattened into a d_p-dimensional vector.

Being a kinematic rigid body, the manipulator's end effector is compactly represented by a 7D vector consisting of its 3D position and orientation represented by a 4D quaternion, although some DoFs may be disabled in certain scenes. For each task, this representation results in an $N_m \times d_m$ matrix encoding the full states of manipulators, where N_m is the number of manipulators needed in the task and $d_m = 3$ or 7 depending on whether rotation is needed. Regarding the interaction between soft bodies and manipulators, we implement one-way coupling between rigid objects and soft bodies and fix all other physical parameters such as particle's mass and manipulator's friction.

Observation While the particle states fully characterize the soft-body dynamics, its high DoFs are hardly tractable for any planning and control algorithm to work with directly. We thus downsample N_k particles as landmarks and stack their positions and velocities (6D for each landmark) into a matrix of size $N_k \times 6$, which serves as the observation of the particle system. Note that landmarks in the same task have fixed relative locations in the plasticine's initial configuration, leading to a consistent particle observation across different configurations of the task. Combining the particle observation with the manipulator state, we end up having $N_k \times 6 + N_m \times d_m$ elements in the observation vector.

Action At each time step, the agent is instructed to update the linear (and angular when necessary) velocities of the manipulators in a kinematic manner, resulting in an action of size $N_m \times d_a$ where $d_a = 3$ or 6 depending on whether rotations of the manipulators are enabled in the task. For each task, we provide global $A_{\min}, A_{\max} \in \mathbb{R}^{d_a}$, the lower and upper bounds on the action, to stabilize the physics simulation.

Goal and Reward Each task is equipped with a target shape represented by its mass tensor, which is essentially its density field discretized into a regular grid of size N_{grid}^3. At each time step t, we compute the mass tensor of the current soft body S_t. Discretizing both target and current shapes into a grid representation allows us to define their similarity by comparing densities at the same locations, avoiding the challenging problem of matching particle systems or point clouds. The complete definition of our reward function includes a similarity metric as well as two regularizers on the high-level motion of the manipulator: $\mathcal{R} = c_1\mathcal{R}_{\text{mass}} + c_2\mathcal{R}_{\text{dist}} + c_3\mathcal{R}_{\text{grasp}}$, where $\mathcal{R}_{\text{mass}}$ measures the L_1 distance between the two shapes' mass tensors as described above, $\mathcal{R}_{\text{dist}}$ is the dot product of the signed distance field (SDF) of the target shape and the current shape's mass tensor, and $\mathcal{R}_{\text{grasp}}$ encourages the manipulators to be closer to the soft bodies.

3.2 Evaluation Suite

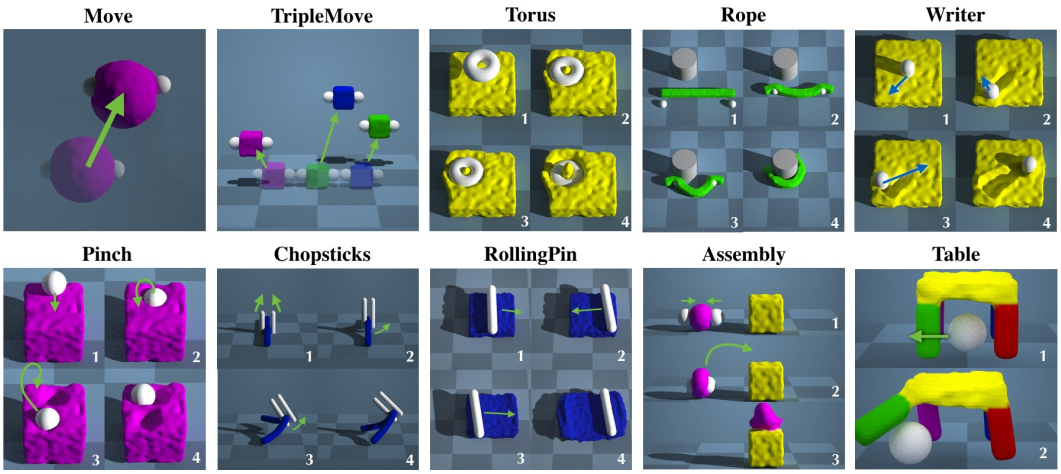


Figure 2: Tasks and reference solutions of PlasticineLab. Certain tasks require multi-stage planning.

PlasticineLab has a diverse collection of 10 tasks (Fig. 2). We describe four representative tasks here, and the remaining six tasks are detailed in Appendix B.

These tasks, along with their variants in different configurations, form an evaluation suite for benchmarking performance of soft-body manipulation algorithms. Each task has 5 variants (50 configurations in total) generated by perturbing the initial and target shapes and the initial locations of manipulators.

Rope The agent needs to wind a rope, modeled as a long plasticine piece, around a rigid pillar with two spherical manipulators. The pillar's position varies in different configurations.

Writer The agent manipulates a "pen" (represented using a vertical capsule), to sculpt a target scribble on cubic plasticine. For each configuration, we generate the scribble by drawing random 2D lines on the plasticine surface. The three-dimensional action controls the tip of the pen.

Chopsticks The agent uses a pair of chopsticks, modeled as two parallel capsules, to pick up the rope on the ground and rotate it into the target location. The manipulator has 7 DoFs: 6 for moving and rotating the pair of chopsticks and 1 for controlling the distance between them.

RollingPin The agent learns to flatten a "pizza dough", which is modeled as a plasticine box, with a rigid rolling pin. We simulate the rolling pin with a 3-DoF capsule: 1) the pin can descend vertically to press the dough; 2) the pin can rotate along the vertical axis to change its orientation; 3) the agent can also roll the pin over the plasticine to flatten it.

4 Differentiable elastoplasticity simulation

The simulator is implemented using Taichi (Hu et al., 2019a) and runs on CUDA. Continuum mechanics is discretized using the Moving Least Squares Material Point Method (MLS-MPM) (Hu et al., 2018), a simpler and more efficient variant of the B-spline Material Point Method (MPM) in computer graphics (Stomakhin et al., 2013). Both Lagrangian particles and Eulerian background grids are used in the simulator. Material properties, include position, velocity, mass, density, and deformation gradients, are stored on Lagrangian particles that move along with the material, while particle interactions and collisions with rigid bodies are handled on the background Eulerian grid. We refer the reader to ChainQueen (Hu et al., 2019b) and DiffTaichi (Hu et al., 2020) for more details on differentiable MPM with elastic materials. Here we focus on extending the material model with *(differentiable) plasticity*, a defining feature of plasticine. We leverage Taichi's reverse-mode automatic differentiation system (Hu et al., 2020) for most of the gradient evaluations.

von Mises yield criterion We use a simple von Mises yield criterion for modeling plasticity, following the work of Gao et al. (2017). According to the von Mises yield criterion, a plasticine particle yields (i.e., deforms plastically) when its second invariant of the deviatoric stress exceeds a certain threshold, and a projection on the deformation gradient is needed since the material "forgets" its rest state. This process is typically called *return mapping* in MPM literature.

Return mapping and its gradients Following Klár et al. (2016) and Gao et al. (2017), we implement the return mapping as a 3D projection process on the singular values of the deformation gradients of each particle. This means we need a singular value decomposition (SVD) process on the particles' deformation gradients, and we provide the pseudocode of this process in Appendix A. For backpropagation, we need to evaluate *gradients of SVD*. Taichi's internal SVD algorithm (McAdams et al., 2011) is iterative, which is numerically unstable when automatically differentiated in a brute-force manner. We use the approach in Townsend (2016) to differentiate the SVD. For zeros appearing in the denominator when singular values are not distinct, we follow Jiang et al. (2016) to push the absolute value of the denominators to be greater than 10^{-6}.

Contact model and its softened version for differentiability We follow standard MPM practices and use grid-base contact treatment with Coulomb friction (see, for example, Stomakhin et al. (2013)) to handle soft body collision with the floor and the rigid body obstacles/manipulators. Rigid bodies are represented as time-varying SDFs. In classical MPM, contact treatments induce a drastic non-smooth change of velocities along the rigid-soft interface. To improve reward smoothness and gradient quality, we use a *softened* contact model during backpropagation. For any grid point, the simulator computes its signed distance d to the rigid bodies. We then compute a smooth *collision strength factor* $s = \min\{\exp(-\alpha d), 1\}$, which increases exponentially when d decays until $d = 0$.

Intuitively, collision effects get stronger when rigid bodies get closer to the grid point. The positive parameter α determines the sharpness of the softened contact model. We linearly blend the grid point velocity before and after collision projection using factor s, leading to a smooth transition zone around the boundary and improved contact gradients.

5 EXPERIMENTS

5.1 EVALUATION METRICS

We first generate five configurations for each task, resulting in 50 different reinforcement learning configurations. We compute the normalized incremental IoU score to measure if the state reaches the goal. We apply the soft IoU (Rahman & Wang, 2016) to estimate the distance between a state and the goal. We first extract the grid mass tensors S, the masses on all grids. Each value S^{xyz} stores how many materials located in the grid point (x, y, z), which is always non-negative. Let two states' 3D mass tensors be S_1 and S_2. We first divide each tensor by their maximum magnitude to normalize its values to $[0, 1]$: $\bar{S}_1 = \frac{S_1}{\max_{ijk} S_1^{ijk}}$ and $\bar{S}_2 = \frac{S_2}{\max_{ijk} S_2^{ijk}}$. Then the softened IoU of the two state is calculated as $IoU(S_1, S_2) = \frac{\sum_{ijk} \bar{S}_1 \bar{S}_2}{\sum_{ijk} \bar{S}_1 + \bar{S}_2 - \bar{S}_1 \bar{S}_2}$. We refer readers to Appendix F for a better explanation for the soft IoU. The final normalized incremental IoU score measures how much IoU increases at the end of the episode than the initial state. For the initial state S_0, the last state S_t at the end of the episode, and the goal state S_g, the normalized incremental IoU score is defined as $\frac{IoU(S_t,S_g)-IoU(S_0,S_g)}{1-IoU(S_0,S_g)}$. For each task, we evaluate the algorithms on five configurations and report an algebraic average score.

5.2 EVALUATIONS ON REINFORCEMENT LEARNING

We evaluate the performance of the existing RL algorithms on our tasks. We use three SOTA model-free reinforcement learning algorithms: Soft Actor-Critic (SAC) (Haarnoja et al., 2017), Twin Delayed DDPG (TD3) (Fujimoto et al., 2018), and Policy Proximal Optimization (PPO) (Schulman et al., 2017). We train each algorithm on each configuration for 10000 episodes, with 50 environment steps per episode.

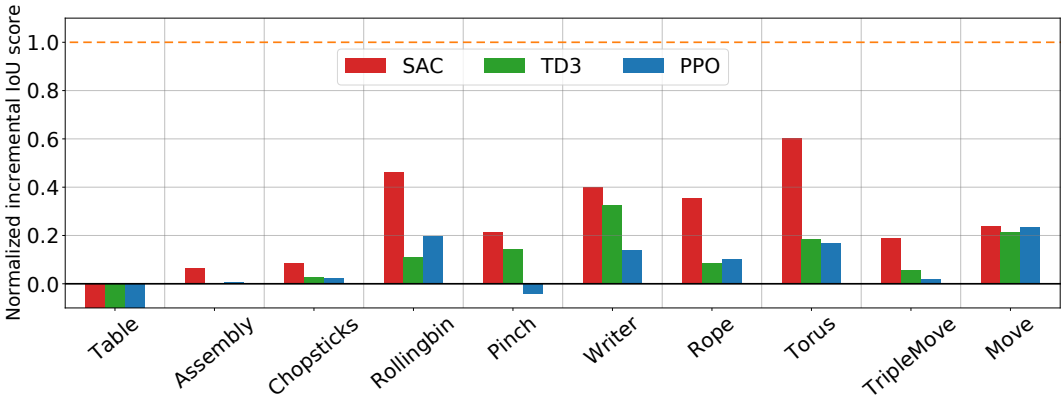


Figure 3: The final normalized incremental IoU score achieved by RL methods within 10^4 epochs. Scores lower than -0.1 are clamped. The dashed orange line indicates the theoretical upper limit.

Figure 3 shows the normalized incremental IoU scores of the tested reinforcement learning algorithms on each scene. Most RL algorithms can learn reasonable policies for **Move**. However, RL algorithms can hardly match the goal shape exactly, which causes a small defect on the final shape matching. We notice that it is common for the RL agent to release the objects during exploration, leading to a free-fall of plasticine under gravity. Then it becomes challenging for the agent to re-grasp the plasticine. This leads to training instability and produces unsatisfactory results. In **Torus**, both TD3 and SAC can find a suitable direction to press the manipulator, but SAC learns a more

stable policy, which yields higher performance. In **Rope**, although the SAC agent fails to move the rope around the pillar, it manages to push the rope towards the pillar and overlap the plasticine with the goal volume, in order to gain partial rewards. SAC also finds the policy to go back and forth to flatten the dough in **RollingPin**. However, those agents usually cannot match the shape exactly, resulting in low normalized incremental IoU scores. These empirical results may have indicated the challenges of solving these geometrical problems accurately. We speculate that Multi-layer Perception Critic Networks might not capture the detailed shape variations well.

In some harder tasks, like **Chopsticks** that requires the agent to handle the 3D rotation carefully, and **Writer** that requires the agent to plan for complex trajectories for carving the traces, the tested algorithm seldom finds a reasonable solution within the limited time (10^4 epochs). In **Assembly**, all agents are stuck in local minima. They move the spherical plasticine closer to the destination but fail to lift it up to achieve an ideal IoU. We think that carefully designed reward shaping might be beneficial in these tasks. In summary, plasticity, together with the soft bodies' high DoFs, poses new challenges for RL algorithms.

5.3 Evaluations on Differentiable Physics For Trajectory Optimization

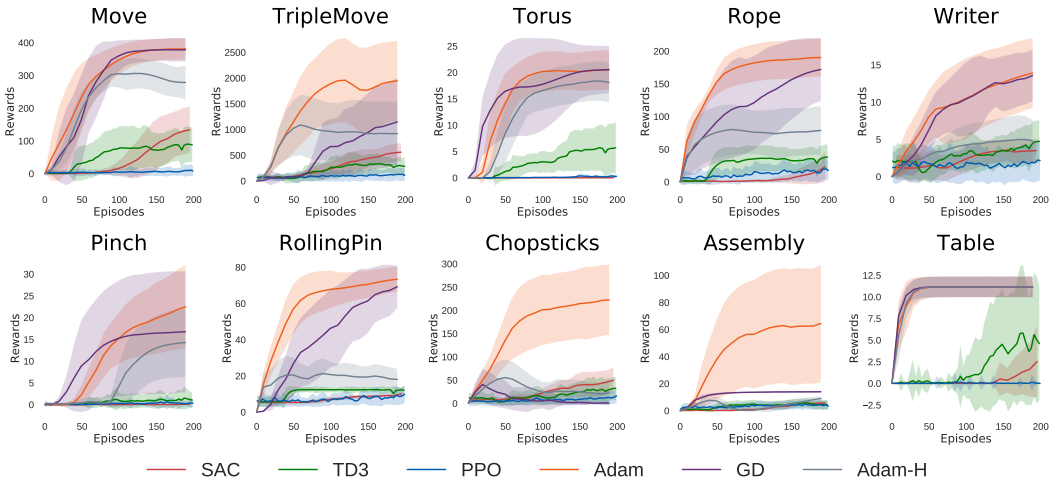


Figure 4: Rewards and their variances in each task w.r.t. the number of episodes spent on training. We clamp the reward to be greater than 0 for a better illustration.

Thanks to the built-in differentiable physics engine in PlasticineLab, we can apply gradient-based optimization to plan open-loop action sequences for our tasks. In gradient-based optimization, for a certain configuration starting at state s, we initialize a random action sequence $\{a_1, \ldots, a_T\}$. The simulator will simulate the whole trajectory, accumulate the reward at each time step, and do back-propagation to compute the gradients of all actions. We then apply a gradient-based optimization method to maximize the sum of rewards. We assume all information of the environment is known. This approach's goal is not to find a controller that can be executed in the real world. Instead, we hope that differentiable physics can help find a solution efficiently and pave roads for other control or reinforcement/imitation learning algorithms.

In Figure 4, we demonstrate the optimization efficiency of differentiable physics by plotting the reward curve w.r.t. the number of environment episodes and compare different variants of gradient descent. We test the Adam optimizer (Adam) and gradient descent with Momentum (Momentum). We use the soft contact model to compute the gradients. We then compare with the Adam optimizer with a hard contact model (Adam-H). For each optimizer, we modestly choose a learning rate of 0.1 or 0.01 per task to handle the different reward scales across tasks. Notice that we only use the soft contact model for computing the gradients and search for a solution. We evaluate all solutions using the environments with hard contacts. In Figure 4, we additionally plot the training curve of reinforcement learning algorithms to demonstrate the efficiency of gradient-based optimization. Results show that optimization-based methods can find a solution for challenging tasks within tens of iterations. Adam outperforms GD in most tasks . This may be attributed to Adam's adaptive

Env	Move	Tri. Move	Torus	Rope	Writer
SAC	$\mathbf{0.24 \pm 0.12}$	$\mathbf{0.19 \pm 0.07}$	$\mathbf{0.60 \pm 0.39}$	$\mathbf{0.35 \pm 0.14}$	$\mathbf{0.40 \pm 0.21}$
TD3	0.21 ± 0.17	0.06 ± 0.10	0.18 ± 0.54	0.08 ± 0.10	0.33 ± 0.28
PPO	0.23 ± 0.21	0.02 ± 0.02	0.17 ± 0.21	0.10 ± 0.06	0.14 ± 0.19
Adam	$\mathbf{0.98 \pm 0.02}$	$\mathbf{0.49 \pm 0.13}$	0.88 ± 0.21	$\mathbf{0.66 \pm 0.16}$	$\mathbf{0.67 \pm 0.24}$
GD	$\mathbf{0.98 \pm 0.02}$	0.31 ± 0.17	$\mathbf{0.90 \pm 0.21}$	0.58 ± 0.16	0.66 ± 0.24
Adam-H	0.94 ± 0.05	0.31 ± 0.13	0.80 ± 0.28	0.46 ± 0.12	0.41 ± 0.30

Env	Pinch	RollingPin	Chopsticks	Assembly	Table
SAC	$\mathbf{0.21 \pm 0.23}$	$\mathbf{0.46 \pm 0.21}$	$\mathbf{0.08 \pm 0.05}$	$\mathbf{0.06 \pm 0.12}$	$\mathbf{-1.52 \pm 3.43}$
TD3	0.14 ± 0.21	0.11 ± 0.02	0.03 ± 0.05	-0.00 ± 0.01	-1.68 ± 3.25
PPO	-0.04 ± 0.15	0.20 ± 0.06	0.02 ± 0.02	0.01 ± 0.00	-5.66 ± 5.18
Adam	$\mathbf{0.49 \pm 0.22}$	$\mathbf{0.92 \pm 0.06}$	$\mathbf{0.71 \pm 0.19}$	$\mathbf{0.67 \pm 0.40}$	0.27 ± 0.35
GD	0.49 ± 0.34	0.88 ± 0.10	0.21 ± 0.13	0.01 ± 0.00	0.00 ± 0.01
Adam-H	0.23 ± 0.13	0.42 ± 0.10	0.20 ± 0.10	0.01 ± 0.00	$\mathbf{0.39 \pm 0.26}$

Table 1: The averaged normalized incremental IoU scores and the standard deviations of each method. Adam-H stands for optimizing on the hard contact model with Adam optimizer. We train RL agent for 10000 episodes and optimizing for 200 episodes for gradient-based approaches.

learning rate scaling property, which fits better for the complex loss surface of the high-dimensional physical process. The hard contact model (Adam-H) performs worse than the soft version (Adam) in most tasks, which validates the intuition that a soft model is generally easier to optimize.

Table 1 lists the normalized incremental IoU scores, together with the standard deviations, of all approaches. The full knowledge of the model provides differentiable physics a chance to achieve more precious results. Gradient descent with Adam can find the way to move the rope around the pillar in **Rope**, jump over the sub-optimal solution in **Assembly** to put the sphere above the box and use the chopsticks to pick up the rope. Even for **Move**, it often achieves better performance by better aligning with the target shape and more stable optimization process.

Some tasks are still challenging for gradient-based approaches. In **TripleMove**, the optimizer minimizes the particles' distance to the closet target shape, which usually causes two or three plasticines to crowd together into one of the target locations. It is not easy for the gradient-based approaches, which have no exploration, to jump out such local minima. The optimizer also fails on the tasks that require multi-stage policies, e.g., **Pinch** and **Writer**. In **Pinch**, the manipulator needs to press the objects, release, and press again. However, after the first touch of the manipulator and the plasticine, any local perturbation of the spherical manipulator doesn't increase the reward immediately, and the optimizer idles at the end.

6 Potential Research Problems to Study using PlasticineLab

Our environment opens ample research opportunities for learning-based soft-body manipulation. Our experiments show that differential physics allows gradient-based trajectory optimization to solve simple planning tasks extremely fast, because gradients provide strong and clear guidance to improve the policy. However, gradients will vanish if the tasks involve detachment and reattachment between the manipulators and the plasticine. When we fail to use gradient-based optimization that is based on local perturbation analysis, we may consider those methods that allow multi-step exploration and collect cumulative rewards, e.g., random search and reinforcement learning. Therefore, it is interesting to study how differentiable physics may be combined with these sampling-based methods to solve planning problems for soft-body manipulation.

Beyond the planning problem, it is also interesting to study how we shall design and learn effective controllers for soft-body manipulation in this environment. Experimental results (Sec. 5.2) indicate that there is adequate room for improved controller design and optimization. Possible directions include designing better reward functions for RL and investigating proper 3D deep neural network structures to capture soft-body dynamics.

A third interesting direction is to transfer the trained policy in PlasticineLab to the real world. While this problem is largely unexplored, we believe our simulator can help in various ways: 1. As shown in Gaume et al. (2018), MPM simulation results can accurately match the real world. In the future,

we may use our simulator to plan a high-level trajectory for complex tasks and then combine with low-level controllers to execute the plan. 2. Our differential simulator can compute the gradient to physical parameters and optimize parameters to fit the data, which might help to close the sim2real gap. 3. PlasticineLab can also combine domain randomization and other sim2real methods (Matas et al., 2018). One can customize physical parameters and the image renderer to implement domain randomization in our simulator. We hope our simulator can serve as a good tool to study real-world soft-body manipulation problems.

Finally, generalization is an important exploration direction. Our platform supports procedure generation and can generate and simulate various configurations with different objects, evaluating different algorithms' generalizability. PlasticineLab is a good platform to design rich goal-condition tasks, and we hope it can inspire future work.

7 Conclusion and Future Work

We presented PlasticineLab, a new differentiable physics benchmark for soft-body manipulation. To the best of our knowledge, PlasticineLab is the first skill-learning environment that simulates elastoplastic materials while being differentiable. The rich task coverage of PlasticineLab allows us to systematically study the behaviors of state-of-the-art RL and gradient-based algorithms, providing clues to future work that combines the two families of methods.

We also plan to extend the benchmark with more articulation systems, such as virtual shadow hands[2]. As a principled simulation method that originated from the computational physics community (Sulsky et al., 1995), MPM is convergent under refinement and has its own accuracy advantages. However, modeling errors are inevitable in virtual environments. Fortunately, apart from serving as a strong supervision signal for planning, the simulation gradient information can also guide systematic identification. This may allow robotics researchers to "optimize" tasks themselves, potentially simultaneously with controller optimization, so that sim-to-real gaps are automatically minimized.

We believe PlasticineLab can significantly lower the barrier of future research on soft-body manipulation skill learning, and will make its unique contributions to the machine learning community.

References

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pp. 4502–4510, 2016. 2

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016. 2

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013. 1

James M Bern, Pol Banzet, Roi Poranne, and Stelian Coros. Trajectory optimization for cable-driven soft robot locomotion. In *Robotics: Science and Systems*, 2019. 3

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 1, 2, 3

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

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016. 2

Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, pp. 7178–7189, 2018. 2, 3

[2] https://en.wikipedia.org/wiki/Shadow_Hand

Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *arXiv preprint arXiv:1611.01652*, 2016. 2

Cosimo Della Santina, Robert K Katzschmann, Antonio Biechi, and Daniela Rus. Dynamic control of soft robots interacting with the environment. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 46–53. IEEE, 2018. 3

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016. 2

Tom Erez and Emanuel Todorov. Trajectory optimization for domains with contacts using inverse dynamics. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4914–4919. IEEE, 2012. 3

Philipp Foehn, Davide Falanga, Naveen Kuppuswamy, Russ Tedrake, and Davide Scaramuzza. Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload. *Robotics: Science and System*, 2017. 3

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1582–1591, 2018. 6

Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020. 2

Ming Gao, Andre Pradhana Tampubolon, Chenfanfu Jiang, and Eftychios Sifakis. An adaptive generalized interpolation material point method for simulating elastoplastic materials. *ACM Transactions on Graphics (TOG)*, 36(6):1–12, 2017. 5, 13

Johan Gaume, T Gast, J Teran, A van Herwijnen, and C Jiang. Dynamic anticrack propagation in snow. *Nature communications*, 9(1):1–10, 2018. 8

Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. Add: Analytically differentiable dynamics for multi-body systems with frictional contact. *arXiv preprint arXiv:2007.00987*, 2020. 3

Thomas George Thuruthel, Yasmin Ansari, Egidio Falotico, and Cecilia Laschi. Control strategies for soft robotic manipulators: A survey. *Soft robotics*, 5(2):149–163, 2018. 3

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2017. 6

Eric Heiden, David Millard, Hejia Zhang, and Gaurav S Sukhatme. Interactive differentiable simulation. *arXiv preprint arXiv:1905.10706*, 2019. 3

Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *International Conference on Learning Representations*, 2020. 3

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 (TOG)*, 37(4):1–14, 2018. 2, 5

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019a. 2, 5

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6265–6271. IEEE, 2019b. 3, 5

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *ICLR*, 2020. 2, 3, 5

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 34(4):1–10, 2015. 4

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, pp. 1–52. 2016. 5

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):1–12, 2016. 5

Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017. 2

Yinxiao Li, Yonghao Yue, Danfei Xu, Eitan Grinspun, and Peter K Allen. Folding deformable objects using predictive simulation and trajectory optimization. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6000–6006. IEEE, 2015. 3

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018. 2

Junbang Liang, Ming C Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. *Advances in Neural Information Processing Systems*, 2019. 3

Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. Fluid directed rigid body control using deep reinforcement learning. *ACM Trans. Graph.*, 37(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201334. URL https://doi.org/10.1145/3197517.3201334. 3

Andrew D Marchese, Russ Tedrake, and Daniela Rus. Dynamics and trajectory optimization for a soft spatial fluidic elastomer manipulator. *The International Journal of Robotics Research*, 35(8): 1000–1019, 2016. 3

Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018. 9

Aleka McAdams, Andrew Selle, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. Computing the singular value decomposition of 3x3 matrices with minimal branching and elementary floating point operations. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2011. 5

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 preprint arXiv:1806.08047*, 2018. 2

PhysX. *"PhysX SDK"*. URL "https://developer.nvidia.com/physx-sdk". 2

Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014. 3

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8494–8502, 2018. 2

Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C Lin. Scalable differentiable physics for learning and control. *arXiv preprint arXiv:2007.02168*, 2020. 3

Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing*, pp. 234–244. Springer, 2016. 6

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. *ICCV*, 2019. 2

Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. *arXiv preprint arXiv:1806.06094*, 2018. 2

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 6

Koushil Sreenath, Nathan Michael, and Vijay Kumar. Trajectory generation and control of a quadrotor with a cable-suspended load-a differentially-flat hybrid system. In *2013 IEEE International Conference on Robotics and Automation*, pp. 4888–4895. IEEE, 2013. 3

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):1–10, 2013. 5

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. 9

Sarah Tang and Vijay Kumar. Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2216–2222. IEEE, 2015. 3

Russ Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832). http://underactuated.mit.edu/, 2020. Downloaded on 2020-09-30. 3

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. 1, 2

James Townsend. Differentiating the singular value decomposition. Technical report, Technical Report 2016, https://j-towns. github. io/papers/svd-derivative . . . , 2016. 5

Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *CVPR*, pp. 9068–9079, 2018. 2

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11097–11107, 2020. 2

Mengyuan Yan, Yilin Zhu, Ning Jin, and Jeannette Bohg. Self-supervised learning of state estimation for manipulating deformable linear objects. *IEEE Robotics and Automation Letters*, 5(2): 2372–2379, 2020. 3

A SIMULATOR IMPLEMENTATION DETAILS

von Mises plasticity return mapping pseudo code Here we list the implementation of the forward return mapping (Gao et al., 2017). Note the SVD in the beginning leads to gradient issues that need special treatments during backpropagation.

```
def von_Mises_return_mapping(F):
    # F is the deformation gradient before return mapping

    U, sig, V = ti.svd(F)

    epsilon = ti.Vector([ti.log(sig[0, 0]), ti.log(sig[1, 1])])
    epsilon_hat = epsilon - (epsilon.sum() / 2)
    epsilon_hat_norm = epsilon_hat.norm()
    delta_gamma = epsilon_hat_norm - yield_stress / (2 * mu)

    if delta_gamma > 0: # Yields!
        epsilon -= (delta_gamma / epsilon_hat_norm) * epsilon_hat
        sig = make_matrix_from_diag(ti.exp(epsilon))
        F = U @ sig @ V.transpose()
    return F
```

Parameters We use a yield stress of 50 for plasticine in all tasks except **Rope**, where we use 200 as the yield stress to prevent the rope from fracturing. We use $\alpha = 666.7$ in the soft contact model.

Parallelism and performance Our parallel mechanism is based on ChainQueen (Hu et al.,2019b). A single MPM substep involves three stages: particle to grid transform (p2g), grid boundary conditions (gridop), and grid to particle transform (g2p). In each stage, we use a parallel for-loop to loop over all particles (p2g) or grids (for gridop and g2p) to do physical computations and ap-ply atomic operators to write results back to particles/grids. Gradient computation, which needs to reverse the three stages, is automatically done by DiffTaich (Hu et al., 2020).

We benchmark our simulator's performance on each scene in Table 2. Note one step of our environment has 19 MPM substeps.

Env	Forward	Forward + Backward
Table	19.14 ms (52 FPS)	38.18 ms (26 FPS)
Assembly	19.22 ms (52 FPS)	40.35 ms (25 FPS)
Chopsticks	19.53 ms (51 FPS)	39.36 ms (25 FPS)
RollingPin	19.13 ms (52 FPS)	38.07 ms (26 FPS)
Pinch	19.08 ms (52 FPS)	37.85 ms (26 FPS)
Writer	19.09 ms (52 FPS)	37.61 ms (27 FPS)
Rope	19.47 ms (51 FPS)	43.76 ms (23 FPS)
Torus	19.02 ms (53 FPS)	37.67 ms (27 FPS)
Tri.Move	20.91 ms (48 FPS)	49.11 ms (20 FPS)
Move	19.08 ms (52 FPS)	39.43 ms (25 FPS)

Table 2: Performance on an NVIDIA GTX 1080 Ti GPU. We show the average running time for a single forward or forward + backpropagation step for each scene.

B MORE DETAILS ON THE EVALUATION SUITE

Move The agent uses two spherical manipulators to grasp the plasticine and move it to the target location. Each manipulator has 3 DoFs controlling its position only, resulting in a 6D action space.

TripleMove The agent operates three pairs of spherical grippers to relocate three plasticine boxes into the target positions. The action space has a dimension of 18. This task is challenging to both RL and gradient-based methods.

Torus A piece of cubic plasticine is fixed on the ground. In each configuration of the task, we generate the target shape by randomly relocating the plasticine and push a torus mold towards it. The agent needs to figure out the correct location to push down the mold.

Pinch In this task, the agent manipulates a rigid sphere to create dents on the plasticine box. The target shape is generated by colliding the sphere into the plasticine from random angles. To solve this task, the agent needs to discover the random motion of the sphere.

Assembly A spherical piece of plasticine is placed on the ground. The agent first deforms the sphere into a target shape and then moves it onto a block of plasticine. The manipulators are two spheres.

Table This task comes with a plasticine table with four legs. The agent pushes one of the table legs towards a target position using a spherical manipulator.

C Reinforcement Learning Setup

We use the open-source implementation of SAC, PPO and TD3 in our environments. We list part of the hyperparameters in Table 3 for SAC, Table 5 for TD3 and Table 4 for PPO. We fix $c_1 = 10, c_2 = 10$ and $c_3 = 1$ for all environments' reward.

Table 3: SAC Parameters

gamma	0.99
policy lr	0.0003
entropy lr	0.0003
target update coef	0.0003
batch size	256
memory size	1000000
start steps	1000

Table 4: PPO Parameters

update steps	2048
lr	0.0003
entropy coef	0
value loss coef	0.5
batch size	32

Table 5: TD3 Parameters

start timesteps	1000
batch size	256
gamma	0.99
tau	0.0055
policy noise	0.2
noise clip	0.5

D Ablation Study on Yield Stress

To study the effects of yield stress, we run experiments on a simple **Move** configuration (where SAC can solve it well) with different yield stress. We vary the yield stress from 10 to 1000 to generate 6 environments and train SAC on them. Figure 5 plots the agents' performances w.r.t. the number of training episodes. The agents achieve higher reward as the yield stress increase, especially in the beginning. Agents in high yield stress environments learn faster than those in lower yield stress environments. We attribute this to the smaller plastic deformation in higher yield stress environments. If we train the agents for more iterations, those in environments with yield stress larger than 100 usually converge to a same performance level, close to solving the problem. However, materials in environments with yield stress smaller than 100 tend to deform plastically, making it hard to grasp and move an object while not destroying its structure. This demonstrates a correlation between yield stress and task difficulty.

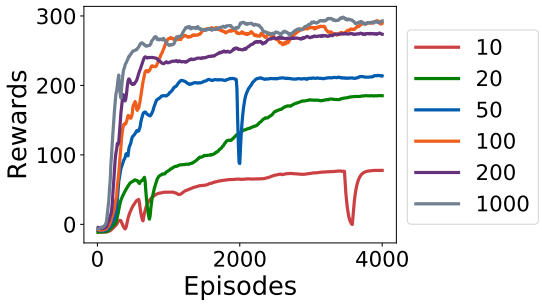


Figure 5: Rewards w.r.t. the number of training episode on 6 environments. Their yield stresses are 10, 20, 50, 100, 200, and 1000.

E DETAILED RESULTS ON ALL CONFIGURATIONS

We run each algorithm with 3 random seeds and each scene has 5 configurations. Therefore, $3 \times 5 = 15$ random seeds in total are used for each algorithm on each scene. We report the normalized incremental IoU scores and the standard deviations for each configuration in Table 6. We also show the optimization efficiency of differentiable physics in Figure 6.

Env \ Method	SAC	TD3	PPO	Adam	GD	Adam-H
Move-v1	0.25 ± 0.05	$\mathbf{0.30 \pm 0.09}$	0.00 ± 0.00	0.98 ± 0.00	$\mathbf{0.98 \pm 0.00}$	0.87 ± 0.04
Move-v2	$\mathbf{0.43 \pm 0.07}$	0.41 ± 0.04	0.43 ± 0.07	0.96 ± 0.01	$\mathbf{0.98 \pm 0.02}$	0.95 ± 0.00
Move-v3	0.20 ± 0.01	0.00 ± 0.00	$\mathbf{0.30 \pm 0.15}$	0.98 ± 0.00	0.98 ± 0.01	$\mathbf{1.00 \pm 0.00}$
Move-v4	0.07 ± 0.03	0.16 ± 0.12	$\mathbf{0.37 \pm 0.20}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.97 ± 0.02
Move-v5	$\mathbf{0.23 \pm 0.01}$	0.20 ± 0.14	0.06 ± 0.05	$\mathbf{1.00 \pm 0.00}$	0.97 ± 0.02	0.92 ± 0.02
Tri.Move-v1	$\mathbf{0.20 \pm 0.08}$	0.00 ± 0.00	0.02 ± 0.01	$\mathbf{0.47 \pm 0.14}$	0.24 ± 0.02	0.19 ± 0.03
Tri.Move-v2	$\mathbf{0.19 \pm 0.04}$	0.08 ± 0.06	0.01 ± 0.01	$\mathbf{0.55 \pm 0.05}$	0.27 ± 0.02	0.29 ± 0.06
Tri.Move-v3	$\mathbf{0.17 \pm 0.02}$	0.05 ± 0.06	0.01 ± 0.01	$\mathbf{0.42 \pm 0.09}$	0.13 ± 0.05	0.22 ± 0.01
Tri.Move-v4	$\mathbf{0.16 \pm 0.03}$	0.00 ± 0.00	0.00 ± 0.01	$\mathbf{0.38 \pm 0.12}$	0.31 ± 0.05	0.35 ± 0.10
Tri.Move-v5	$\mathbf{0.23 \pm 0.10}$	0.16 ± 0.15	0.05 ± 0.00	$\mathbf{0.61 \pm 0.03}$	0.59 ± 0.15	0.52 ± 0.04
Torus-v1	$\mathbf{0.45 \pm 0.07}$	-0.10 ± 0.14	0.07 ± 0.01	0.94 ± 0.01	$\mathbf{0.95 \pm 0.00}$	0.93 ± 0.01
Torus-v2	0.17 ± 0.22	$\mathbf{0.29 \pm 0.37}$	0.04 ± 0.01	0.47 ± 0.01	$\mathbf{0.55 \pm 0.25}$	0.26 ± 0.11
Torus-v3	$\mathbf{0.99 \pm 0.01}$	-0.04 ± 0.94	0.53 ± 0.20	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
Torus-v4	$\mathbf{0.40 \pm 0.38}$	0.23 ± 0.18	0.12 ± 0.06	1.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.92 ± 0.06
Torus-v5	$\mathbf{1.00 \pm 0.00}$	0.54 ± 0.37	0.08 ± 0.06	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.91 ± 0.09
Rope-v1	$\mathbf{0.37 \pm 0.08}$	0.16 ± 0.08	0.09 ± 0.04	$\mathbf{0.62 \pm 0.02}$	0.61 ± 0.03	0.39 ± 0.01
Rope-v2	$\mathbf{0.39 \pm 0.05}$	0.07 ± 0.10	0.15 ± 0.01	$\mathbf{0.83 \pm 0.00}$	0.64 ± 0.09	0.49 ± 0.01
Rope-v3	$\mathbf{0.46 \pm 0.01}$	0.14 ± 0.10	0.16 ± 0.03	0.62 ± 0.01	$\mathbf{0.74 \pm 0.06}$	0.39 ± 0.06
Rope-v4	$\mathbf{0.12 \pm 0.09}$	0.00 ± 0.00	0.01 ± 0.01	$\mathbf{0.41 \pm 0.01}$	0.32 ± 0.04	0.35 ± 0.00
Rope-v5	$\mathbf{0.42 \pm 0.05}$	0.05 ± 0.07	0.10 ± 0.03	$\mathbf{0.83 \pm 0.00}$	0.60 ± 0.09	0.68 ± 0.00
Writer-v1	0.49 ± 0.10	$\mathbf{0.62 \pm 0.27}$	0.26 ± 0.09	0.77 ± 0.02	$\mathbf{0.98 \pm 0.03}$	0.71 ± 0.41
Writer-v2	$\mathbf{0.66 \pm 0.03}$	0.41 ± 0.28	0.18 ± 0.17	$\mathbf{1.00 \pm 0.00}$	0.72 ± 0.20	0.42 ± 0.23
Writer-v3	$\mathbf{0.36 \pm 0.09}$	0.13 ± 0.17	0.07 ± 0.31	0.53 ± 0.28	$\mathbf{0.60 \pm 0.26}$	0.26 ± 0.10
Writer-v4	0.16 ± 0.12	$\mathbf{0.21 \pm 0.15}$	0.04 ± 0.06	$\mathbf{0.61 \pm 0.03}$	0.52 ± 0.03	0.13 ± 0.08
Writer-v5	$\mathbf{0.33 \pm 0.23}$	0.26 ± 0.23	0.14 ± 0.11	0.41 ± 0.01	0.47 ± 0.11	$\mathbf{0.52 \pm 0.02}$
Pinch-v1	$\mathbf{0.17 \pm 0.13}$	0.01 ± 0.00	0.02 ± 0.02	$\mathbf{0.60 \pm 0.01}$	0.59 ± 0.02	0.23 ± 0.00
Pinch-v2	$\mathbf{0.41 \pm 0.13}$	0.18 ± 0.23	0.00 ± 0.02	$\mathbf{0.71 \pm 0.05}$	0.21 ± 0.25	0.02 ± 0.00
Pinch-v3	$\mathbf{0.20 \pm 0.04}$	0.13 ± 0.17	-0.15 ± 0.26	$\mathbf{0.68 \pm 0.06}$	0.21 ± 0.04	0.33 ± 0.00
Pinch-v4	$\mathbf{0.37 \pm 0.25}$	0.29 ± 0.30	-0.09 ± 0.14	0.16 ± 0.01	$\mathbf{0.43 \pm 0.29}$	0.17 ± 0.00
Pinch-v5	-0.09 ± 0.03	$\mathbf{0.11 \pm 0.10}$	0.01 ± 0.01	0.30 ± 0.03	$\mathbf{1.00 \pm 0.00}$	0.41 ± 0.00
RollingPin-v1	$\mathbf{0.31 \pm 0.01}$	0.10 ± 0.00	0.17 ± 0.06	$\mathbf{0.88 \pm 0.05}$	0.87 ± 0.04	0.46 ± 0.09
RollingPin-v2	$\mathbf{0.44 \pm 0.10}$	0.08 ± 0.01	0.18 ± 0.05	$\mathbf{0.86 \pm 0.02}$	0.78 ± 0.15	0.40 ± 0.08
RollingPin-v3	$\mathbf{0.62 \pm 0.17}$	0.11 ± 0.00	0.19 ± 0.06	$\mathbf{1.00 \pm 0.00}$	0.98 ± 0.03	0.51 ± 0.03
RollingPin-v4	$\mathbf{0.48 \pm 0.27}$	0.14 ± 0.03	0.20 ± 0.06	$\mathbf{0.87 \pm 0.04}$	0.84 ± 0.04	0.37 ± 0.04
RollingPin-v5	$\mathbf{0.47 \pm 0.26}$	0.11 ± 0.00	0.25 ± 0.03	$\mathbf{0.96 \pm 0.04}$	0.92 ± 0.07	0.33 ± 0.10
Chopsticks-v1	$\mathbf{0.05 \pm 0.04}$	-0.00 ± 0.00	0.04 ± 0.02	$\mathbf{0.80 \pm 0.04}$	0.18 ± 0.03	0.31 ± 0.15
Chopsticks-v2	$\mathbf{0.14 \pm 0.02}$	0.07 ± 0.07	0.02 ± 0.02	$\mathbf{0.95 \pm 0.01}$	0.32 ± 0.17	0.24 ± 0.02
Chopsticks-v3	$\mathbf{0.10 \pm 0.07}$	0.03 ± 0.04	-0.01 ± 0.01	$\mathbf{0.60 \pm 0.18}$	0.23 ± 0.09	0.15 ± 0.03
Chopsticks-v4	$\mathbf{0.07 \pm 0.04}$	0.04 ± 0.06	0.03 ± 0.01	$\mathbf{0.69 \pm 0.06}$	0.16 ± 0.13	0.13 ± 0.05
Chopsticks-v5	$\mathbf{0.05 \pm 0.02}$	0.00 ± 0.00	0.02 ± 0.01	$\mathbf{0.52 \pm 0.18}$	0.14 ± 0.06	0.18 ± 0.09
Assembly-v1	$\mathbf{0.06 \pm 0.07}$	-0.00 ± 0.00	0.00 ± 0.00	$\mathbf{0.90 \pm 0.05}$	0.01 ± 0.00	0.01 ± 0.00
Assembly-v2	$\mathbf{0.01 \pm 0.01}$	-0.01 ± 0.01	0.00 ± 0.00	$\mathbf{0.61 \pm 0.43}$	0.01 ± 0.00	0.01 ± 0.00
Assembly-v3	$\mathbf{0.12 \pm 0.17}$	0.00 ± 0.00	0.00 ± 0.00	$\mathbf{0.62 \pm 0.43}$	0.00 ± 0.00	0.00 ± 0.00
Assembly-v4	$\mathbf{0.11 \pm 0.15}$	0.00 ± 0.00	0.01 ± 0.00	$\mathbf{0.62 \pm 0.43}$	0.01 ± 0.00	0.00 ± 0.00
Assembly-v5	$\mathbf{0.01 \pm 0.00}$	0.00 ± 0.00	0.01 ± 0.00	$\mathbf{0.59 \pm 0.41}$	0.00 ± 0.00	0.01 ± 0.00
Table-v1	-2.29 ± 1.92	$\mathbf{-1.16 \pm 1.68}$	-3.66 ± 0.20	$\mathbf{0.90 \pm 0.03}$	-0.01 ± 0.00	0.63 ± 0.22
Table-v2	-0.83 ± 1.24	$\mathbf{-0.14 \pm 0.23}$	-2.19 ± 0.23	0.01 ± 0.00	-0.00 ± 0.00	$\mathbf{0.49 \pm 0.05}$
Table-v3	$\mathbf{-4.30 \pm 6.33}$	-5.89 ± 4.80	-15.61 ± 0.57	0.02 ± 0.00	0.02 ± 0.00	$\mathbf{0.26 \pm 0.29}$
Table-v4	$\mathbf{0.27 \pm 0.01}$	-1.20 ± 1.86	-5.49 ± 0.43	-0.01 ± 0.00	-0.01 ± 0.00	$\mathbf{0.08 \pm 0.12}$
Table-v5	-0.46 ± 0.67	$\mathbf{-0.02 \pm 0.05}$	-1.36 ± 0.25	0.41 ± 0.00	0.01 ± 0.00	$\mathbf{0.48 \pm 0.07}$

Table 6: The normalized incremental IoU scores of each method on each configuration. We run each experiment with 3 random seeds and report their averages and standard deviations (mean $\pm$ std)

.

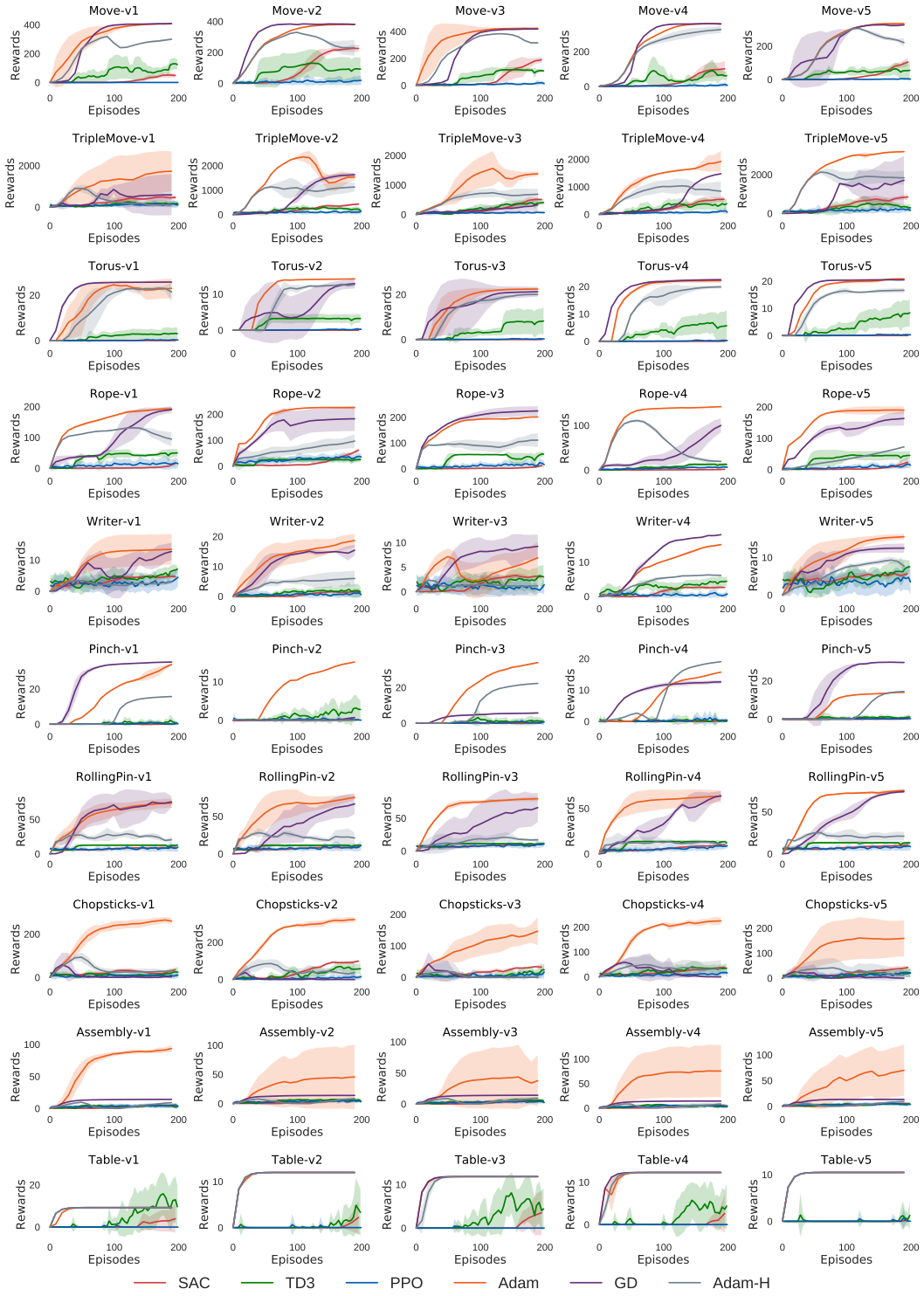


Figure 6: Rewards and variances in each configuration w.r.t. the number of episodes spent on training. We clamp the reward to be greater than 0 for a better illustration.

F EXPLANATION OF SOFT IOU

Let a, b be two numbers and can only be 0 or 1, then $ab = 1$ if and only if $a = b = 1$; $a + b - ab = 1$ if and only if at least one of a, b is 1. If the two mass tensors only contain value of 0 or 1, $\sum S_1 S_2$ equals the number of grids that have value 1 in both tensors, i.e., the "Intersection". For the same reason $\sum S_1 + S_2 - S_1 S_2$ counts the grids that $S_1 = 1$ or $S_2 = 1$, the "Union". Thus, the formula $\sum S_1 S_2 / \sum S_1 + S_2 - S_1 S_2$ computes the standard Intersection over Union (IoU). In our case, we assume the normalized mass tensor S_1 and S_2 are two tensors with positive values. Therefore, we first normalize them so that their values are between 0 and 1, then apply the previous formula to compute a "soft IoU," which approximately describes if two 3D shapes match.