∇*Sim*: Differentiable Physics and Rendering Engines for Parameter Estimation from Video

Anonymous Author(s)
Affiliation
Address
email

Abstract

What we see in a video is governed by the underlying 3D structure of the scene being captured, as well as the physical properties that determine its dynamics. Several of these properties are difficult to estimate from video without explicitly modeling motion dynamics. In this paper, we tackle the ill-posed problem of reconstructing physical and geometric properties of objects from video, using *differentiable simulation*. To this end, we present ∇*Sim* ("gradSim"), a versatile end-to-end differentiable simulator. ∇*Sim* enables backpropagation from video pixels to the physical and geometric parameters which generate them. We derive a framework for general physical simulation that supports a broad range of differentiable models, including 3D rigid bodies, solid and thin-shell deformable bodies, and incompressible fluids. Compared to existing frameworks, ∇*Sim* is tightly integrated with PyTorch, which facilitates interoperability with existing learning modules. Furthermore, all physical simulations can be differentiably rendered to generate videos, enabling the estimation of physical attributes and control parameters from video supervision alone. We show that maintaining a unified computation graph across the physics and rendering engines enables learning of physical attributes, geometry, and dynamics from video. We show that ∇*Sim* can *solve* a variety of inverse simulation and visual model-predictive control tasks, without relying on *state-based* supervision such as object velocities/positions.

1 Introduction

Accurately predicting and simulating future states in an environment, from sparse observations, is a long-standing and challenging goal. Doing so facilitate short- and long-term reasoning and relies fundamentally on an understanding of the generative processes that lead to complex observations. Imagine watching a short video of a basketball bouncing off the ground; can you approximately infer the weight, shape, and elasticity properties of the ball, as well as collision events with the ground? Based on our understanding of physics and image formation, we can make informed predictions about the movement of the ball and perhaps even subsequent observations.

While the ability to infer the *physical properties* of objects from videos is intuitive for humans, computer vision systems still struggle. A central challenge is the insufficient integration of knowledge of the underlying physical laws of motion, the geometry of scene objects, and the rendering processes that generate the observed video sequence.

When modeling physics, state representation is an important design consideration. Recent work [1, 2, 3, 4, 5] expresses system state as a learned joint embedding of the mechanics of interacting objects and the visual representations that lead to the observed images. Such representations, however, lack interpretability due to the entanglement of several factors of variation [6], or simply the lack of structure in unsupervised learning [7, 8]. Our work forgoes any such learned representation of the system state, instead more directly addressing the question: "*can we make accurate estimates of a*

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

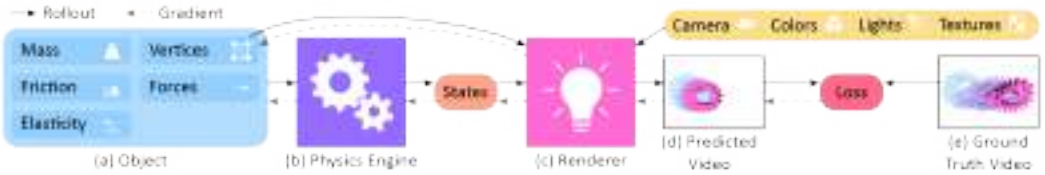


Figure 1: ∇***Sim*:** Given video observations of an evolving physical system (e), we randomly initialize scene object properties (a) and evolve them over time using a differentiable physics engine (b), which generates *states*. Our renderer (c) processes states, object vertices and global rendering parameters to produce image frames for computing our loss. We back-propagate to object shape and physical properties to refine our estimates.

scene's physical parameters from video observations if we explicitly model the laws of motion and the image formation processes that underly our observations"[1]?

Explicitly modeling the dynamics underlying video formation is challenging, even with access to the full system state. This long-standing problem has been treated in the vision, graphics and physics communities [9], leading to the development of robust forward simulation models and algorithms. These simulators are not immediately amenable to solving *inference* problems, due in part to their non-differentiability. As such, they are traditionally used as black-box *forward* processes in this context, often necessitating some surrogate gradient-estimation to enable learning. Examples include finite differencing or REINFORCE [10], which do not scale to high-dimensional parameter spaces. Despite recent progress in likelihood-free inference for black-box forward simulators [11, 12, 13, 14, 15, 16, 17], limitations in terms of data efficiency and scalability to high dimensional parameter spaces persist. We note that incorporating the rendering process when estimating physical parameters of objects is helpful, as it imposes additional constraints that can be leveraged to more accurately infer the objects' geometries. Traditional approaches rely on shading cues [18], whereas more recent *differentiable rendering* methods increase flexibility. Differentiable renderers have, however, thus far been treated independently from (differentiable) physics simulators.

We present ∇*Sim*, a versatile end-to-end differentiable simulator that combines *both* physics and rendering. ∇*Sim* enables backpropagation [19] from video pixels to the time-evoling physical and geometric properties of objects in a scene. We estimate these parameters solely from video observations, and for a wide range of scenes including interactions between rigid, soft and thin-shell bodies, as well as soft robots and incompressible fluids. ∇*Sim* enables data efficient parameter estimation by exploiting the structure of a unified treatment of the physics and the rendering processes, a direction absent in existing differentiable simulator works. Moreover, ∇*Sim* relies on unlabeled video data (albeit synthetic), which is abundant, without requiring expensive sensing equipment.

Our main technical contributions are:

- an integrated simulator that combines differentiable physics and rendering engines,
- the ability to recover many physical properties exclusively from video observations, including friction, elasticity, deformable material parameters, and
- careful and efficient integration with PyTorch, facilitating interoperability with existing machine learning modules and imperative programming.

We evaluate and demonstrate ∇*Sim*'s effectiveness on parameter estimation tasks for rigid, thin-shell and deformable bodies: our estimates are much more accurate than current physics-only differentiable simulators. Additionally, we use our estimated physical parameters to make accurate *predictions* – both of the fine-scale physical states and properties, and of future image frames – in response to applied/external controls. We further employ our image-based predictions to accurate vision-based model-predictive control (MPC) tasks for rigid and challenging deformable bodies, like cloth.

2 Related work

Differentiable physics simulators have seen significant attention and activity, with efforts centered around embedding physics structure into autodifferentiation frameworks. This has enabled differentiation through contact and friction models [20, 21, 22, 23, 24, 25, 26], latent

[1] *Physics* refers to the laws governing the motion and deformation of objects over time, and *rendering* refers to the interaction of these scene elements – including their material properties – with scene lighting when forming image sequences observed by a virtual camera. *Simulator* refers to the combination of these two processes.

state models [27, 28, 29, 30], volumetric soft bodies [31, 32, 33, 34], as well as particle dynamics [28, 35, 36, 34]. In contrast, ∇*Sim* addresses a superset of simulation scenarios, by coupling the physics simulator with a differentiable rendering pipeline. It also supports tetrahedral FEM-based hyperelasticity models to simulate deformable solids and thin-shells.

Recent work on **physics-based deep learning** injects structure in the latent representation space of the dynamics through the Lagrangian and Hamiltonian functions [7, 37, 38, 39, 8, 40], through other conserved quantities [41], or through ground truth supervision [42].

Predicting the effects of forces applied to an object based on a history of sensor observations has also been examined in works related to **models of learned** [43, 44] **and intuitive physics** [45, 46, 47, 48, 49, 50, 51, 52, 53, 54]. This line of work also includes learning models of multi-object interactions [1, 2, 3, 4, 5, 55]. In many cases, intuitive physics approaches are limited in terms of the prediction horizons and complexity of scenes they can handle because they do not model the 3D geometry and object properties sufficiently accurately. **System identification** based on parameterized physics models [56, 57, 58, 59, 60, 61, 62] and inverse simulation [63] are closely related areas.

There is a rich literature on **neural image synthesis**, but we focus on methods that model the 3D scene structure, including voxels [64, 65, 66, 67], meshes [68, 69, 70, 71], and implicit shapes [72, 73, 74, 75, 76, 77]. Generative models sample the 3D geometry to condition the rendering process on it [78]. Latent factors determining 3D structure have also been learned in generative models [6, 79]. Additionally, works that explicitly model radiance fields [80, 81], also make use of geometry by estimating the pose of the camera using structure from motion techniques, such as [82]. In previous work many of these representations have become easy to manipulate through software frameworks like Kaolin [83] and Open3D [84].

Differentiable rendering formulates the image formation process so as to allow for image gradients to be computed w.r.t. the scene geometry, camera, and lighting inputs. Variants based on the rasterization paradigm (NMR, OpenDR, SoftRas) [85, 86, 87] blur the edges of scene triangles prior to image projection, to remove discontinuities in the rendering signal. DIB-R [88] applies this idea to background pixels and proposes an interpolation-based rasterizer for foreground pixels. More sophisticated differentiable renderers can treat physically-based light transport processes [89, 90]. These methods rely instead on the ray tracing, and more readily support higher-order effects such as shadows, secondary light bounces, and global illumination.

3 ∇*Sim*: A unified treatment of differentiable physics and rendering

Traditionally, physics simulation and rendering have been treated as disjoint, mutually exclusive tasks. In this work, we take on a unified view of *simulation* in general, to mean physics simulation *and* rendering. Formally, we define simulation as a function $\text{Sim} : \mathbb{R}^P \times [0, 1] \mapsto \mathbb{R}^H \times \mathbb{R}^W; \text{Sim}(\mathbf{p}, t) = \mathcal{I}$. Here $\mathbf{p} \in \mathbb{R}^P$ is a vector representing the combined state and parameters of the simulation (objects, their physical properties, their geometries, etc.), t denotes the time of simulation (which is conveniently reparameterized to be in the interval $[0, 1]$. Given initial conditions $\mathbf{p}_0$, the simulation function, at each timestep t, produces an image $\mathcal{I}$ of height H and width W. If this function Sim were differentiable, then the gradient of a function $\text{Sim}(\mathbf{p}, t)$ with respect to the simulation parameters $\mathbf{p}$ intuitively tells us that *perturbing* $\mathbf{p}$ *by an infinitesimal* $\delta\mathbf{p}$ *will change the output of the simulation from* $\mathcal{I}$ *to* $\mathcal{I} + \nabla\textit{Sim}(\mathbf{p}, t)\delta\mathbf{p}$. Such a construct would enable any gradient-based optimization routine to infer physical parameters from video, by defining a *loss function* over image space $l(\mathcal{I}, .)$, and by descending this loss landscape along a direction parallel to $-\nabla\text{Sim}(.)$. To realise this, we turn to the paradigms of *computational graphs* and *differentiable programming*.

∇*Sim* comprises two broad components: a *differentiable physics engine* that computes the physical states of the world at each time instant, and a *differentiable renderer* that renders the world to a 2D image. Contrary to existing differentiable physics [20, 21, 22, 23, 24, 25, 26, 34] or differentiable rendering [85, 86, 87, 88] approaches, we adopt a holistic view and construct a computational graph spanning them both. We describe each of the components in greater detail.

3.1 Differentiable physics engine

Under Lagrangian mechanics, the state of a physical system can be described in terms of generalized coordinates $\mathbf{q}$, generalized velocities $\dot{\mathbf{q}} = \mathbf{u}$, and design, or model parameters θ. For the purposes of exposition, we make no distinction between rigid-bodies, deformable solids, or thin-shell models of

cloth, etc. Although the specific choices of coordinates and parameters vary, the simulation procedure is virtually unchanged. We denote the combined state vector by $\mathbf{s}(t) = [\mathbf{q}(t), \mathbf{u}(t)]$.

The dynamic evolution of the system is governed by a second order differentiable equations (ODE) of the form $\mathbf{M}\ddot{\mathbf{s}} = \mathbf{f}(\mathbf{s})$, where $\mathbf{M}$ is a mass matrix that may also depend on our state and design parameters θ. Solutions to ODEs of this type may be obtained through black box numerical integration methods, and their derivatives calculated through the continuous adjoint method [91]. However, we instead consider our physics engine as a differentiable operation that provides an implicit relationship between a state vector $\mathbf{s}^- = \mathbf{s}(t)$ at the start of a time step, and the updated state at the end of the time step $\mathbf{s}^+ = \mathbf{s}(t + \Delta t)$. An arbitrary discrete time integration scheme can be then be abstracted as the function $\mathbf{g}(\mathbf{s}^-, \mathbf{s}^+, \theta) = \mathbf{0}$, relating the initial and final system state and the model parameters θ. By the implicit function theorem, if we can specify a loss function l at the output of the simulator, we can compute $\frac{\partial l}{\partial \mathbf{s}^-}$ as $\mathbf{c}^T \frac{\partial \mathbf{g}}{\partial \mathbf{s}^-}$, where $\mathbf{c}$ is the solution to the linear system $\frac{\partial \mathbf{g}}{\partial \mathbf{s}^+}^T \mathbf{c} = -\frac{\partial l}{\partial \mathbf{s}^+}^T$, and likewise for the model parameters θ.

While the partial derivatives $\frac{\partial \mathbf{g}}{\partial \mathbf{s}^-}$, $\frac{\partial \mathbf{g}}{\partial \mathbf{s}^+}$, $\frac{\partial \mathbf{g}}{\partial \theta}$ can be computed by graph-based auto-differentation frameworks [92, 93, 94], program transformation approaches such as DiffTaichi, and Google Tangent [34, 95] are particularly well-suited to simulation code. We use an embedded subset of Python syntax, which computes the adjoint of each simulation kernel at runtime, and generates C++/CUDA [96] code. Kernels are wrapped as custom autograd operations on PyTorch tensors, which allows users to focus on the definition of physical models, and leverage the PyTorch tape-based autodiff to track the overall program flow. While this formulation is general enough to represent explicit, multi-step, or fully implicit time-integration schemes, we employ semi-implicit Euler integration, which is the preferred integration scheme for most simulators [97].

3.2 Physical models

We now discuss some of the physical models available in ∇*Sim*.

Deformable Solids: As opposed to existing simulators that use grid-based methods for differentiable soft-body simulation [31, 34], we adopt a finite element (FEM) model with constant strain tetrahedral elements common in computer graphics [98]. We use the stable NeoHookean constitutive model of Smith et al. [99] that derives per-element forces from the following strain energy density:

$$\Psi(\mathbf{q}, \theta) = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - \alpha)^2 - \frac{\mu}{2}\log(I_C + 1), \tag{1}$$

where I_C, J are invariants of strain, $\theta = [\mu, \lambda]$ are the Lamé parameters, and α is a per-element actuation value that allows the element to expand and contract.

Numerically integrating the energy density over each tetrahedral mesh element with volume V_e gives the total elastic potential energy, $U(\mathbf{q}, \theta) = \sum V_e \Psi_e$. The forces due to this potential $\mathbf{f}_e(\mathbf{s}, \theta) = -\nabla_{\mathbf{q}} U(\mathbf{q}, \theta)$, can computed analytically, and their gradients are obtained using the adjoint method (*cf.* Section 3.1).

Deformable Thin-Shells: To model thin-shells such as clothing, we use constant strain triangular elements embedded in 3D. The Neo-Hookean constitutive model above is applied to model in-plane elastic deformation, with the addition of a bending energy $\mathbf{f}_b(\mathbf{s}, \theta) = k_b \sin(\frac{\phi}{2} + \alpha)\mathbf{d}$, where k_b is the bending stiffness, ϕ is the dihedral angle between two triangular faces, α is a per-edge actuation value that allows the mesh to flex inwards or outwards, and $\mathbf{d}$ is the force direction given by [100]. We also include a lift/drag model that approximates the effect of the surrounding air on the surface of mesh.

Rigid Bodies: We represent the state of a 3D rigid body as $\mathbf{q}_b = [\mathbf{x}, \mathbf{r}]$ consisting of a position $\mathbf{x} \in \mathbb{R}^3$, and a quaternion $\mathbf{r} \in \mathbb{R}^4$. The generalized velocity of a body is $\mathbf{u}_b = [\mathbf{v}, \omega]$ and the dynamics of each body is given by the Newton-Euler equations,

$$\begin{bmatrix} m & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \tau \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ \omega \times \mathbf{I}\omega \end{bmatrix} \tag{2}$$

where the mass m and inertia matrix $\mathbf{I}$ (expressed at the center of mass) are considered design parameters θ.

Contact: We adopt a compliant contact model that associates elastic and damping forces with each nodal contact point. The model is parameterized by four scalars $\theta = [k_e, k_d, k_f, \mu]$, corresponding

to elastic stiffness, damping, frictional stiffness, and friction coefficient respectively. To prevent interpenetration we use a proportional penalty-based force, $\mathbf{f}_n(\mathbf{s}, \theta) = -\mathbf{n}[k_e C(\mathbf{q}) + k_d \dot{C}(\mathbf{u})]$, where $\mathbf{n}$ is a contact normal, and C is a gap function measure of overlap projected on to $\mathbb{R}^+$. We model friction using a relaxed Coulomb model [101] $\mathbf{f}_f(\mathbf{s}, \theta) = -\mathbf{D}[\min(\mu|\mathbf{f}_n|, k_f \mathbf{u}_s)]$, where $\mathbf{D}$ is a basis of the contact plane, and $\mathbf{u}_s = \mathbf{D}^T\mathbf{u}$ is the sliding velocity at the contact point. While these forces are only C^0 continuous, we found that this was sufficient for optimization over a variety of objectives.

More physical simulations: We also implement a number of other differentiable simulations such as pendula, mass-springs, and incompressible fluids [102]. We defer this discussion to the supplementary material, as such systems have already been demonstrated in prior art.

3.3 Differentiable rendering engine

A renderer expects *scene description* inputs and generates color image outputs, all according to a sequence of image formation stages defined by the *forward* graphics pipeline. The scene description includes a complete *geometric* descriptor of scene elements, their associated material/reflectance properties, light source definitions, and virtual camera parameters. The rendering process is not generally differentiable, as *visibility* and *occlusion* events introduce discontinuities. Most interactive renderers, e.g. for real-time applications, employ a *rasterization* process to project 3D geometric primitives onto 2D pixel coordinates, resolving these visibility events with non-differentiable operations. Our experiments employ two differentiable alternatives to traditional rasterization, SoftRas [87] and DIB-R [88], both of which rely on smoothing triangle edges by replacing their discontinuities with sigmoids. This has the effect of blurring triangle edges into semi-transparent boundaries, thereby removing the non-differentiable discontinuity of traditional rasterization. DIB-R distinguishes between *foreground pixels* (associated to the principal object being rendered in the scene) and *background pixels* (for all other objects, if any). The latter are rendered using the same technique as SoftRas while the former are rendered by bilinearly sampling a texture using differentiable UV coordinates. Given its efficiency benefits, we rely preferentially on DIB-R whenever rendering speed becomes a bottleneck.

∇*Sim* performs differentiable physics and rendering at independent and adjustable rates, allowing us to trade computation for accuracy by rendering fewer frames than physics simulation updates.

4 Experiments

We conducted multiple experiments to test the efficacy of ∇*Sim*, which we summarize in this section. Our experiments broadly span two categories: *physical parameter estimation from video*, and *vision-based model-predictive control (MPC)*. Each of our experiments comprises an *environment* $\mathcal{E}$ that applies a particular set of physical forces/constraints, a (differentiable) *loss function* $\mathcal{L}$ that implicitly specifies an objective, and an *initial guess* θ_0 of the state of the simulation. The goal is to recover optimal parameters θ^* that minimize $\mathcal{L}$, by backpropagating through the simulator.

4.1 Physical parameter estimation from video

In the first suite of experiments, we assess the capabilities of ∇*Sim* to accurately infer a variety of physical properties such as mass, restitution, elasticity, stiffness parameters, initial conditions, and external forces from image/video observations. To the best of our knowledge, ours is the first study to infer a fine-grained parameter set from video observations. We implement a set of competitive baselines that often use strictly more information on the task, to test the efficacy of ∇*Sim*.

4.1.1 Rigid bodies (`rigid`)

Our first environment (codenamed `rigid`) deals with the estimation of physical and material properties of rigid objects from videos. We curate a dataset of 14 meshes, comprising primitive shapes such as boxes, cones, cylinders, as well as non-convex shapes from ShapeNet [103] and DexNet [104]. With uniformly sampled initial dimensions, poses, velocities, and physical properties (density, elasticity, and friction parameters), we apply a*known* impulse to the object, and record a video of the resultant trajectory. Inference with ∇*Sim* is done by picking an initial guess of the mass (at random), unrolling a *differentiable* simulation using this guess, comparing the rendered out video with the true video (pixelwise mean-squared error - MSE), and updating the initial guess by gradient descent over the unrolled computation graph.

Approach	Mean abs. err. (kg)	Abs. Rel. err.
Average	0.2022	0.1031
Random	0.2653	0.1344
ConvNet baseline	0.13	0.0094
REINFORCEd PyBullet	0.0928	0.3668
diffphysics (3D supervision)	1.35e-9	5.17e-9
∇*Sim*	2.36e-5	9.01e-5

Table 1: **Mass estimation**: ∇*Sim* obtains *precise* mass estimates, comparing favourably even with approaches that require 3D supervision (*diffphysics*). We report the mean abolute error and absolute relative errors for all approaches evaluated.

	mass	elasticity		friction	
Approach	m	k_d	k_e	k_f	μ
Average	1.7713	3.7145	2.3410	4.1157	0.4463
Random	10.0007	4.18	2.5454	5.0241	0.5558
ConvNet baseline	0.029	0.14	0.14	0.17	0.096
diffphy. (3D sup.)	1.70e-8	0.036	0.0020	0.0007	0.0107
∇*Sim*	2.87e-4	0.4	0.0026	0.0017	0.0073

Table 2: **Rigid-body parameter estimation**: ∇*Sim* estimates contact parameters (elasticity, friction) to a high degree of accuracy, despite estimating them from video. Diffphy. requires accurate 3D ground-truth at 30 fps. We report absolute *relative* errors for each approach evaluated.

Table 1 shows the results for an initial control experiment, designed to evaluate the ability to predict the mass of an object from video, and knowing the impulse applied to it. We compare ∇*Sim* with three other baselines: ∇PyBullet [105], *diffphysics*, and a ConvNet baseline. For the ConvNet baseline, we use EfficientNet (b-0) [106] and resize frames to 64×64,features are extracted from efficientnet (b-0) and further reduced to feature maps of size $4 \times 4 \times 32$. Features are then concatenated for all frames and fed to an MLP with 4 linear layers and ReLU activations. The output layer is linear. The network is trained with a MSE loss. The *diffphysics* baseline is a strict subset of ∇*Sim*, that only inolves the differentiable physics engine. Thus, it needs precise 3D states as supervision, as opposed to ∇*Sim*, which requires only a rendered out video. This strong supervision is the primary factor for the superior performance of *diffphysics*. However, ∇*Sim* is able to very precisely estimate mass from video, to a relative abolute error of 9.01e-5, which is nearly 2 orders of magnitude better than the ConvNet baseline. The “Average" an “Random" baselines predict the dataset mean and a uniformly sampled random number from the domain respectively.

To investigate whether analytical *differentiability* is required, we also evaluate the performance of a ∇PyBullet baseline, and apply a black-box gradient estimation technique ([10]), similar to [45]. We find this baseline particularly sensitive to several simulation parameters, and worse-performing compared to the ConvNet baseline.

In Table 2, we also estimate the parameters of our compliant contact model from video observations alone. The trend is similar to Table 1, and ∇*Sim* is able to precisely recover the parameters of the simulation.

4.1.2 Correlation between image (MSE) difference and physical parameter error

With several complex differentiable components (i.e.: the physics engine and the renderer) chained together, one might wonder if such a combination of highly nonlinear functions can provide meaningful gradients to guide physical parameter estimation. To explore this adversarial line of thought, we analyze the loss landscape of pixelwise mean-squared error. Fig. 2 depicts the loss landscape when optimizing for the mass of a rigid body whose shape, elastic, and frictional properties are known. We take an object with unit mass (1 kg), and examine the behavior of mean-squared error over a range of initial masses (0.1 kg to 5 kg). Surprisingly, we find that the loss landscape is quite well-behaved and easy to navigate for momentum-based optimizers. Especially, applying MSE to the first and last frames of the predicted and true video sequences seems to provide the best gradients for this task. This is an indication that ∇*Sim* gradients provide useful information to estimate the mass of an object. This also justifies the superior performance of ∇*Sim* in Table 1.

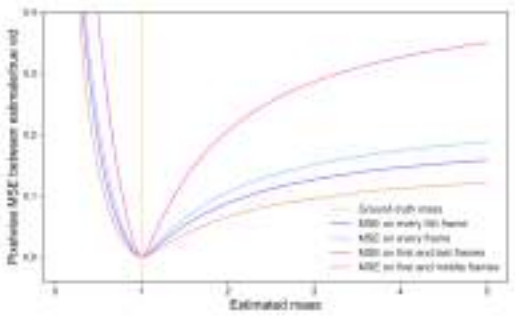


Figure 2: **Loss landscape** when optimizing for the mass of a rigid-body with known shape using ∇*Sim*. Despite images being formed by a highly nonlinear process (simulation), the loss landscape is remarkably smooth, for a range of initialization errors.

	Deformable solid FEM			Thin-shell (cloth)
	Per-particle mass	Material properties		Per-particle velocity
	m	μ	λ	v
Approach	**Rel. MAE**	**Rel. MAE**	**Rel. MAE**	**Rel. MAE**
3D supervision (diffphysics)	0.032	0.0025	0.0024	0.127
∇*Sim*	0.048	0.0054	0.0056	0.026

Table 3: **Parameter estimation of deformable objects**: We estimate per-particle masses and material properties (for solid def. objects) and per-particle velocities for cloth. In the case of cloth, there is a perceivable performance drop in *diffphysics*, as the center of mass of a cloth is often outside the body, which results in ambiguity.

4.1.3 Parameter estimation for non-rigid objects (`deformable`)

One core strength / novelty of ∇*Sim* is its ability to handle a rich class of deformable objects, such as deformable solids and thin-shells (cloth). We conduct a series of experiments to investigate the ability of ∇*Sim* to recover physical parameters of deformable solids and thin-shell solids (cloth). In Table 3 we demonstrate the ability of ∇*Sim* to accurately recover the parameters of 100 instances of deformable objects (cloth, bouncing spheres, beams).

4.2 Vision-based model-predictive control (MPC)

To investigate whether the gradients computed by ∇*Sim* are meaningful for vision-based tasks, we conduct a range of *visual model-predictive control* (MPC) experiments, involving the actuation of deformable objects towards a *visual* target pose. While traditional, *state-based* MPC requires a goal specification in *state*-space, *visual* MPC specifies a goal by means of an image of the desired end-configuration.

4.2.1 Deformable solids (`control-walker`, `control-fem`)

The first example (`control-walker`) involves a 2D walker model, originally from DiffTaichi [34]. We represent the walker as a FEM triangle mesh (104 elements), as opposed to a particle grid, and train a neural network (NN) control policy to actuate the walker to reach a target pose on the right-hand side of an image. Our NN consists of one fully connected layer and a $\tanh$ activation. The network input is a set of 8 time-varying sinusoidal signals, and the output is a scalar activation value per-tetrahedron. Each triangle uses elasticity parameters of $\mu = 10^4$, $\lambda = 10^4$, and contact parameters of $k_e = 10^4$, $k_d = 10^3$, $k_f = 10^3$, $k_\mu = 0.5$, and is allowed to change its area by a maximum of 20%, to encourage physically-plausible solutions. We include an activation penalty of $l_{cost} = 0.1\|\alpha\|_2$, where α is the vector of scalar activation values. ∇*Sim* is able to *solve* this environment within 3 iterations of gradient descent, by minimizing a pixelwise MSE between the last frame of the rendered video and the goal image as shown in Fig. 3 (column 2, bottom).

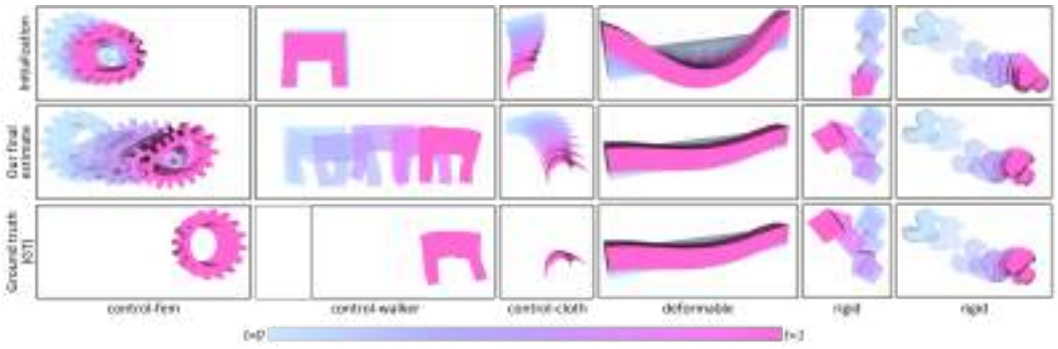


Figure 3: **Qualitative results**: ∇*Sim* accurately estimates physical parameters for diverse, complex environments. For `control-fem` and `control-walker` experiments, we train a neural network to actuate a soft body towards a target *image* (GT). For `control-cloth`, we optimize the cloth's initial velocity to hit a target pose (GT), all in under nonlinear lift/drag forces. For `deformable` experiments, we optimize the material properties of a beam to match a video. In the `rigid` experiments, we estimate contact parameters (elasticity/friction) and object density to match a *video* (GT). We visualize entire time sequences (t) with color-coded blends.

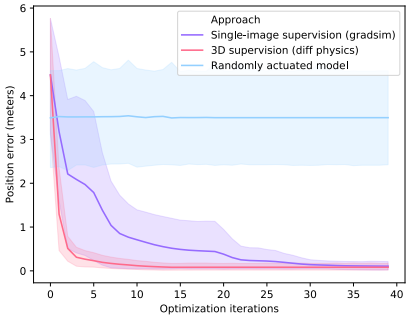
In our second test, we formulate a more challenging 3D control problem (`control-fem`) where the goal is to actuate a soft-body FEM object (a *gear*) consisting of 1152 tetrahedral elements to move to a target position as shown in Fig. 3 (left). We use the same NN architecture as in the 2D walker example, and for visual MPC use the Adam [107] optimizer to minimize our pixelwise MSE metric. To evaluate ∇*Sim* accuracy we train a baseline model (*diffphysics*) that uses strong supervision and minimizes the MSE between the target position and the precise 3D location of the center-of-mass (COM) of the FEM model at each time step (i.e.: a *dense* reward). We test both *diffphysics* and ∇*Sim* against a naive baseline that generates random activations and plot convergence in Fig. 4a.

While *diffphysics* appears to be a strong performer on this task, it is important to note that it uses explicit 3D supervision at each timestep (i.e., 30 fps). In contrast, ∇*Sim* uses a *single image* as an implicit target, and yet manages to achieve the goal state, albeit taking a longer number of iterations.

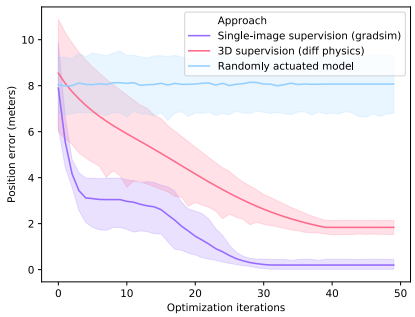
4.2.2 Cloth (`control-cloth`)

We design an experiment to control a piece of cloth by optimizing the initial velocity such that it reaches a pre-specified target. In each *episode*, a random cloth is spawned, comprising between 64 and 2048 triangles, and a new start/goal combination is chosen. Across all episodes, we fix elasticity parameters to $\mu = \lambda = 10^4$.

In this challenging setup, we notice that *state-based* MPC (*diffphysics*) is often unable to accurately reach the target. We believe this is due to the underdetermined nature of the problem, since, for objects such as cloth, the COM by itself does not uniquely determine the configuration of the object. Visual MPC on the otherhand, provides a more well-defined problem. An illustration of the task is presented in Fig. 3 (column 3), and the convergence of the methods shown in Fig. 4b.



(a) Results of various approaches on the **`control-fem`** environment (6 randomseeds; each randomseed corresponds to a different goal configuration). While *diffphysics* performs well, it assumes strong 3D supervision. In contrast, ∇*Sim* is able to *solve* the task by using just a *single image* of the target configuration.



(b) Results on **`control-cloth`** environment (5 randomseeds; each controls the dimensions and initial/target poses of the cloth). *diffphysics* converges to a suboptimal solution due to ambiguity in specifying the pose of a cloth via its center-of-mass. ∇*Sim* *solves* the environment using a single target image.

Figure 4: Convergence of ∇*Sim* visual MPC compared with strongly supervised (diffphysics), and random action policies.

5 Conclusion and Future Work

We presented ∇*Sim*, a versatile differentiable simulator that integrates differentiable physics and rendering engines. We demonstrated the benefits of such a unified approach by estimating physical properties and time-evolving geometry for scenes with complex dynamics and deformations from raw video observations. We also demonstrated the applicability of this efficient and accurate estimation scheme on an end-to-end visuomotor control task. The latter case highlights ∇*Sim*'s efficient integration with PyTorch, facilitating interoperability with existing machine learning modules. An interesting avenue of future work would be to extend our differentiable simulation to contact-rich motion, articulated bodies and higher-fidelity physically-based renderers – doing so would take us closer to operating on videos captured from the real world.

6 Broader impact

Much progress has been made on end-to-end learning in visual domains. If successful, image and video understanding promises far-reaching applications from safer autonomous vehicles to more realistic computer graphics, but relying on these tools for planning and control poses substantial risk.

Neural information processing systems have shown experimentally promising results on visuomotor tasks, yet fail in unpredictable and unintuitive ways when deployed in real-world applications. If embodied learning agents are to play a broader role in the physical world, they must be held to a higher standard of interpretability. Establishing trust requires not just empirical, but explanatory evidence in the form of physically grounded models.

Our work provides a bridge between gradient- and model-based optimization. Explicitly modeling visual dynamics using well-understood physical principles has important advantages for human explainability and debuggability.

Unlike end-to-end neural architectures which distribute bias across a large set of parameters, ∇Sim trades their flexibility for physical interpretability. This does not eliminate the risk of bias in simulation, but allows us to isolate bias to physically grounded variables. Where discrepancy occurs, users can probe the model to obtain end-to-end gradients with respect to variation in physical orientation and material properties, or pixelwise differences. Differentiable simulators like ∇Sim afford a number of opportunities for use and abuse. We envision the following scenarios.

- A technician could query a trained model, "What physical parameters is the steering controller most sensitive to?", or "What happens if friction were slightly lower on that stretch of roadway?"
- An engery-conscious organization could use ∇Sim to accelerate convergence of reinforcement learning models, reducing the energy consumption required for training.
- Using differentiable simulation, an adversary could efficiently construct a physically plausible scene causing the model to produce an incorrect prediction or take an unsafe action.

Video understanding is a world-building exercise with inherent modeling bias. Using physically well-studied models makes those modeling choices explicit, however mitigating the risk of bias still requires active human participation in the modeling process. While a growing number of physically-based rendering and animation engines have been built, our approach requires a high upfront engineering cost in simulation infrastructure. To operationalize these tools, we anticipate practitioners will need to devote significant effort to identifying and replicating unmodeled dynamics from real world-trajectories. Differentiable simulation offers a computationally efficient and physically interpretable pathway for doing so, by estimating physical trajectories and the properties which govern them.

References

[1] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Neural Information Processing Systems*. 2017. 1, 3

[2] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B. Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. In *Robotics Science and Systems*, 2019. 1, 3

[3] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. *CoRR*, abs/1812.10972, 2018. 1, 3

[4] Sébastien Ehrhardt, Aron Monszpart, Niloy J. Mitra, and Andrea Vedaldi. Learning A physical long-term predictor. *CoRR*, abs/1703.00247, 2017. 1, 3

[5] Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *CoRR*, abs/1612.00341, 2016. 1, 3

[6] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016. 1, 3

[7] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks, 2019. 1, 3

[8] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks, 2020. 1, 3

[9] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2016. 2

[10] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, May 1992. 2, 6

[11] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *CoRR*, abs/1906.01728, 2019. 2

[12] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference, 2019. 2

[13] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. Mansinghka. Picture: A probabilistic programming language for scene perception. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4390–4399, 2015. 2

[14] *Causal and compositional generative models in online perception*, London, UK, 07/2017 2017. 2

[15] Ilker Yildirim, Tejas Kulkarni, Winrich Freiwald, and Joshua Tenenbaum. Efficient analysis-by-synthesis in vision: A computational framework, behavioral tests, and comparison with neural representations. 07 2015. 2

[16] Ilker Yildirim, Mario Belledonne, Winrich Freiwald, and Josh Tenenbaum. Efficient inverse graphics in biological face processing. *Science Advances*, 6(10), 2020. 2

[17] Jiajun Wu, Joshua B Tenenbaum, and Pushmeet Kohli. Neural scene de-rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[18] E. Prados and O. Faugeras. *Shape From Shading*, pages 375–388. Springer US, Boston, MA, 2006. 2

[19] Andreas Griewank and Andrea Walther. Introduction to automatic differentiation. *PAMM*, 2(1):45–49, 2003. 2

[20] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. 2, 3

[21] 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 31*, pages 7178–7189. Curran Associates, Inc., 2018. 2, 3

[22] Changkyu Song and Abdeslam Boularias. Learning to slide unknown objects with differentiable physics simulations, 2020. 2, 3

[23] Changkyu Song and Abdeslam Boularias. Identifying mechanical models through differentiable simulations, 2020. 2, 3

[24] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis Wyffels. A differentiable physics engine for deep learning in robotics. *CoRR*, abs/1611.01652, 2016. 2, 3

[25] Jiajun Wu, Erika Lu, Pushmeet Kohli, William T Freeman, and Joshua B Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, 2017. 2, 3

[26] Google Research. *Tiny Differentiable Simulator*, 2020 (accessed May 15, 2020). 2, 3

[27] Vincent Le Guen and Nicolas Thome. Disentangling physical dynamics from unknown factors for unsupervised video prediction, 2020. 3

[28] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks, 2018. 3

[29] Miguel Jaques, Michael Burke, and Timothy M. Hospedales. Physics-as-inverse-graphics: Joint unsupervised learning of objects and physics from video. *CoRR*, abs/1905.11169, 2019. 3

[30] Eric Heiden, David Millard, Hejia Zhang, and Gaurav S. Sukhatme. Interactive differentiable simulation, 2019. 3

[31] 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. *CoRR*, abs/1810.01054, 2018. 3, 4

[32] 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 Trans. Graph.*, 37(4), July 2018. 3

[33] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. In *Neural Information Processing Systems*, pages 771–780, 2019. 3

[34] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *International Conference on Learning Representations*, 2020. 3, 4, 7

[35] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019. 3

[36] Yunzhu Li, Toru Lin, Kexin Yi, Daniel Bear, Daniel L. K. Yamins, Jiajun Wu, Joshua B. Tenenbaum, and Antonio Torralba. Visual grounding of learned physical models, 2020. 3

[37] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. In *International Conference on Learning Representations*, 2020. 3

[38] Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks, 2019. 3

[39] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators, 2019. 3

[40] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control, 2019. 3

[41] Jiajun Wu, Joseph J Lim, Hongyi Zhang, Joshua B Tenenbaum, and William T Freeman. Physics 101: Learning physical object properties from unlabeled videos. In *British Machine Vision Conference*, 2016. 3

[42] Martin Asenov, Michael Burke, Daniel Angelov, Todor Davchev, Kartic Subr, and Subramanian Ramamoorthy. Vid2param: Modelling of dynamics parameters from video, 2019. 3

[43] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards, 2015. 3

[44] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. *CoRR*, abs/1606.02378, 2016. 3

[45] Kiana Ehsani, Shubham Tulsiani, Saurabh Gupta, Ali Farhadi, and Abhinav Gupta. Use the force, luke! learning to predict physical forces by simulating effects. In *CVPR*, 2020. 3, 6

[46] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian image understanding: Unfolding the dynamics of objects in static images. *CoRR*, abs/1511.04048, 2015. 3

[47] Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. "what happens if..." learning to predict the effect of forces in images. *CoRR*, abs/1603.05600, 2016. 3

[48] Abhinav Gupta, Alexei A. Efros, and Martial Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *European Conference on Computer Vision(ECCV)*, 2010. 3

[49] Sébastien Ehrhardt, Aron Monszpart, Niloy J. Mitra, and Andrea Vedaldi. Unsupervised intuitive physics from visual observations. *CoRR*, abs/1805.05086, 2018. 3

[50] L. Yu, N. Duncan, and S. Yeung. Fill and transfer: A simple physics-based approach for containability reasoning. In *IEEE International Conference on Computer Vision (ICCV)*, pages 711–719, 2015. 3

[51] Peter W. Battaglia, Jessica B. Hamrick, and Joshua B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013. 3

[52] Richard Mann, Allan Jepson, and Jeffrey Mark Siskind. The computational perception of scene dynamics. *Computer Vision and Image Understanding*, 65(2):113 – 128, 1997. 3

[53] Carlo Innamorati, Bryan Russell, Danny Kaufman, and Niloy Mitra. Neural re-simulation for generating bounces in single images. pages 8718–8727, 10 2019. 3

[54] Trevor Standley, Ozan Sener, Dawn Chen, and Silvio Savarese. image2mass: Estimating the mass of an object from its image. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 324–333. PMLR, 13–15 Nov 2017. 3

[55] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016. 3

[56] Mathieu Salzmann and Raquel Urtasun. Physically-based motion models for 3d tracking: A convex formulation. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, page 2064–2071, USA, 2011. IEEE Computer Society. 3

[57] Marcus Brubaker, David Fleet, and Aaron Hertzmann. Physics-based person tracking using the anthropomorphic walker. *International Journal of Computer Vision*, 87:140–155, 03 2010. 3

[58] Krzysztof Kozlowski. *Modelling and Identification in Robotics*. Advances in Industrial Control. Springer, London, 1998. 3

[59] P. M. Wensing, S. Kim, and J. E. Slotine. Linear matrix inequalities for physically consistent inertial parameter identification: A statistical perspective on the mass distribution. *IEEE Robotics and Automation Letters*, 3(1):60–67, 2018. 3

[60] Giovanni Sutanto, Austin S. Wang, Yixin Lin, Mustafa Mukadam, Gaurav S. Sukhatme, Akshara Rai, and Franziska Meier. Encoding physical constraints in differentiable newton-euler algorithm, 2020. 3

[61] Kun Wang, Mridul Aanjaneya, and Kostas Bekris. A first principles approach for data-efficient system identification of spring-rod systems via differentiable physics engines, 2020. 3

[62] Bin Wang, Paul G. Kry, Yuanmin Deng, Uri M. Ascher, Hui Huang, and Baoquan Chen. Neural material: Learning elastic constitutive material and damping models from sparse data. *CoRR*, abs/1808.04931, 2018. 3

[63] D.J. Murray-Smith. The inverse simulation approach: a focused review of methods and applications. *Mathematics and Computers in Simulation*, 53(4):239 – 247, 2000. 3

[64] Philipp Henzler, Niloy J. Mitra, and Tobias Ritschel. Escaping plato's cave using adversarial training: 3d shape from unstructured 2d image collections. *CoRR*, abs/1811.11606, 2018. 3

[65] Despoina Paschalidou, Ali Osman Ulusoy, Carolin Schmitt, Luc van Gool, and Andreas Geiger. Raynet: Learning volumetric 3d reconstruction with ray potentials, 2019. 3

[66] Edward Smith, Scott Fujimoto, and David Meger. Multi-view silhouette and depth decomposition for high resolution 3d object representation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6479–6489. Curran Associates, Inc., 2018. 3

[67] Thu Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yong-Liang Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. *CoRR*, abs/1806.06575, 2018. 3

[68] Edward J. Smith, Scott Fujimoto, Adriana Romero, and David Meger. Geometrics: Exploiting geometric structure for graph-encoded objects. *CoRR*, abs/1901.11461, 2019. 3

[69] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images, 2018. 3

[70] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation, 2018. 3

[71] Hassan Abu Alhaija, Siva Karthik Mustikovela, Andreas Geiger, and Carsten Rother. Geometric image synthesis, 2018. 3

[72] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction, 2019. 3

[73] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *CoRR*, abs/1812.02822, 2018. 3

[74] Mateusz Michalkiewicz, Jhony K. Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 3

[75] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision, 2019. 3

[76] Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *CoRR*, abs/1901.05103, 2019. 3

[77] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space, 2018. 3

[78] Yiyi Liao, Katja Schwarz, Lars Mescheder, and Andreas Geiger. Towards unsupervised learning of generative models for 3d controllable image synthesis, 2019. 3

[79] S. M. Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S. Morcos, Marta Garnelo, Avraham Ruderman, Andrei A. Rusu, Ivo Danihelka, Karol Gregor, David P. Reichert, Lars Buesing, Theophane Weber, Oriol Vinyals, Dan Rosenbaum, Neil Rabinowitz, Helen King, Chloe Hillier, Matt Botvinick, Daan Wierstra, Koray Kavukcuoglu, and Demis Hassabis. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 3

[80] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. 3

[81] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 2019. 3

[82] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[83] Krishna Murthy Jatavallabhula, Edward Smith, Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. Kaolin: A pytorch library for accelerating 3d deep learning research, 2019. 3

[84] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 3

[85] Matthew M. Loper and Michael J. Black. Opendr: An approximate differentiable renderer. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 154–169. Springer International Publishing, 2014. 3

[86] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of Computer Vision and Pattern Recognition*, 2018. 3

[87] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *Proceedings of International Conference on Computer Vision*, 2019. 3, 5

[88] Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Neural Information Processing Systems*, 2019. 3, 5

[89] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 37(6):222:1–222:11, 2018. 3

[90] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 38(6), 2019. 3

[91] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Neural Information Processing Systems*, pages 6571–6583, 2018. 4

[92] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Neural Information Processing Systems*, pages 8024–8035. Curran Associates, Inc., 2019. 4

[93] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 4

[94] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. 4

[95] Bart van Merriënboer, Alexander B Wiltschko, and Dan Moldovan. Tangent: automatic differentiation using source code transformation in python. In *arXiv*, 2017. 4

[96] David Kirk et al. Nvidia cuda software and gpu parallel computing architecture. In *ISMM*, volume 7, pages 103–104, 2007. 4

[97] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE, 2015. 4

[98] Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction. In *Acm siggraph 2012 courses*, pages 1–50. 2012. 4

[99] Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Transactions on Graphics (TOG)*, 37(2):1–15, 2018. 4

[100] Robert Bridson, Sebastian Marino, and Ronald Fedkiw. Simulation of clothing with folds and wrinkles. In *ACM SIGGRAPH 2005 Courses*, pages 3–es. 2005. 4

[101] Emanuel Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6054–6061. IEEE, 2014. 5

[102] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999. 5

[103] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 5

[104] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. 2017. 5

[105] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019. 6

[106] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019. 6

[107] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 8

Supplementary Material: ∇*Sim*: Differentiable Physics and Rendering Engines for Parameter Estimation from Video

Anonymous Author(s)
Affiliation
Address
email

A1 Discrete Adjoint Method

In the main paper, we presented a formulation of time-integration using the discrete adjoint method that represents an arbitrary time-stepping scheme through the implicit relation,

$$\mathbf{g}(\mathbf{s}^-, \mathbf{s}^+, \theta) = \mathbf{0}. \tag{3}$$

This formulation is general enough to represent both *explicit* or *implicit* time-stepping methods. While explicit methods are often simple to implement, they may require extremely small time-steps for stability, which is problematic for reverse-mode automatic differentiation frameworks that must explicitly store the input state for each discrete timestep invocation of the integration routine. On the other hand, implicit methods can introduce computational overhead or unwanted numerical dissipation [1]. For this reason, many real-time physics engines employ a semi-implicit (*symplectic*) Euler integration scheme [2], due to its ease of implementation and numerical stability in most meaningful scenarios (conserves energy for systems where the Hamiltonian is time-invariant).

We now give a concrete example of the discrete adjoint method applied to semi-implicit Euler. For the state variables defined above, the integration step may be written as follows,

$$\mathbf{g}(\mathbf{s}^-, \mathbf{s}^+, \theta) = \begin{bmatrix} \mathbf{u}^+ - \mathbf{u}^- - \Delta t \mathbf{M}^{-1} \mathbf{f}(\mathbf{s}^-) \\ \mathbf{q}^+ - \mathbf{q}^- - \Delta t \mathbf{u}^+ \end{bmatrix} = \mathbf{0}. \tag{4}$$

Note that in general, the mass matrix $\mathbf{M}$ is a function of $\mathbf{q}$ and θ. For conciseness we only consider the dependence on θ, although the overall procedure is unchanged in the general case. We provide a brief sketch of computing the gradients of $\mathbf{g}(\mathbf{s}^-, \mathbf{s}^+, \theta)$. In the case of semi-implicit integration above, these are given by the following equations:

$$\frac{\partial \mathbf{g}}{\partial \mathbf{s}^-} = \begin{bmatrix} -\Delta t \mathbf{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{q}(t)} & -\mathbf{I} - \Delta t \mathbf{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{u}(t)} \\ -\mathbf{I} & 0 \end{bmatrix} \quad \frac{\partial \mathbf{g}}{\partial \mathbf{s}^+} = \begin{bmatrix} 0 & \mathbf{I} \\ \mathbf{I} & -\Delta t \mathbf{I} \end{bmatrix} \quad \frac{\partial \mathbf{g}}{\partial \theta} = \begin{bmatrix} -\Delta t \frac{\partial \mathbf{M}^{-1}}{\partial \theta} \\ \mathbf{0} \end{bmatrix}. \tag{5}$$

In the case of semi-implicit Euler, the triangular structure of these Jacobians allows the adjoint variables to be computed explicitly. For fully implicit methods such as backwards Euler, the Jacobians may create a linear system that must be first solved to generate adjoint variables.

A2 Physical Models

We now undertake a more detailed discussion of the physical models implemented in ∇*Sim*.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

(a) Triangular FEM element

(b) Tetrahedral FEM element

Figure 5: **Mesh Discretization**: We use triangular (a) and tetrahedral (b) FEM models with angle-based and volumetric activation parameters, α. These mesh-based discretizations are a natural fit for our differentiable rasterization pipeline, which is designed to operate on triangles.

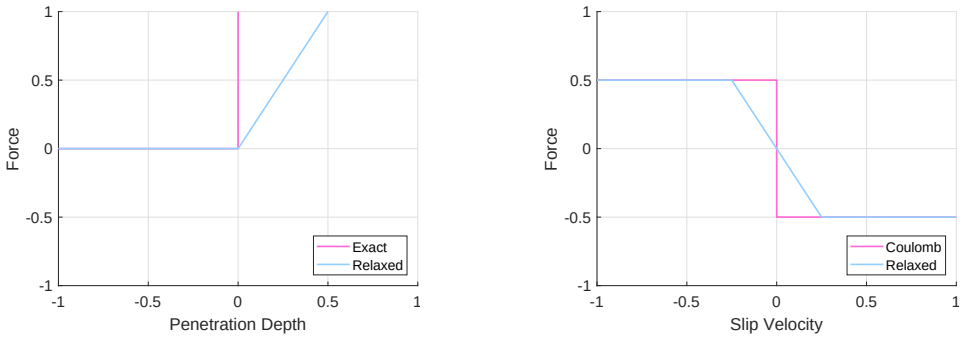
A2.1 Finite element method

As described in section 3.2 ("Physical models"), we use a hyperelastic constitutive model based on the neo-Hookean model of Smith et al. [3]:

$$\Psi(\mathbf{q}, \theta) = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - \alpha)^2 - \frac{\mu}{2}\log(I_C + 1). \tag{6}$$

The Lamé parameters, λ, μ, control the element's resistance to shearing and volumetric strains. These may be specified on a per-element basis, allowing us to represent heterogeneous materials. In contrast to other work using particle-based models [4], we adopt a mesh-based discretization for deformable shells and solids. For thin-shells, such as cloth, the surface is represented by a triangle mesh as in Figure 5a, enabling straightforward integration with our triangle mesh-based differentiable rasterizer [5, 6]. For solids, we use a tetrahedral FEM model as illustrated in Figure 5b. Both these models include a per-element activation parameter α, which for thin-shells, allows us to control the relative dihedral angle between two connected faces. For tetrahedral meshes, this enables changing the element's volume, enabling locomotion, as in the `control-fem` example.

A2.2 Contact



(a) (b)

Figure 6: **Contact Model**: To model non-interpenetration constraints we use a relaxed model of contact that replaces a delta function with a linear hinge corresponding to a quadratic penalty energy (a). To model friction we use a relaxed Coulomb model, that replaces the step function with a symmetric hinge (b).

Implicit contact methods based on linear complementarity formulations (LCP) of contact may be used to maintain hard non-penetration constraints [7]. However, we found relaxed models of contact—used in typical physics engines [2]—were sufficient for our experiments. In this approach, contact forces

are derived from a one-sided quadratic potential, giving rise to penalty forces of the form 6a. While Coulomb friction may also be modeled as an LCP, we use a relaxed model where the *stick* regime is represented by a stiff quadratic potential around the origin, and a linear portion in the *slip* regime, as shown in Figure 6b. To generate contacts, we test each vertex of a mesh against a collision plane and introduce a contact within some threshold distance d.

A2.3 Pendula

We also implement simple and double pendula, as toy examples of well-behaved and chaotic systems respectively, and estimate the parameters of the system (i.e., the length(s) of the rod(s) and initial angular displacement(s)), by comparing the rendered videos (assuming uniformly random initial guesses) with the true videos. As pendula have extensively been studied in the context of differentiable physics simulation [8, 7, 9, 10, 11, 12], we focus on more challenging systems which have not been studied in prior art.

A2.4 Incompressible fluids

As an example of incompressible fluid simulation, we implement a smoke simulator following the popular semi-Lagrangian advection scheme of Stam *et al.* [13]. At 2:20 in our supplementary video attachment, we show an experiment which optimizes the initial velocities of smoke particles to form a desired pattern. Similar schemes have already been realized differentiably, e.g. in DiffTaichi [4] and autograd [14].

A3 Source-code transformation for automatic differentiation

The discrete adjoint method requires computing gradients of physical quantities with respect to state and design parameters. To do so, we adopt a source code transformation approach to perform reverse mode automatic differentiation [15, 16]. We use a domain-specific subset of the Python syntax extended with primitves for representing vectors, matrices, and quaternions. Each type includes functions for acting on them, and the corresponding adjoint method. An example simulation kernel is then defined as follows:

```
@kernel
def integrate_particles(
    x : tensor(float3),
    v : tensor(float3),
    f : tensor(float3),
    w : tensor(float),
    gravity : tensor(float3),
    dt : float,
    x_new : tensor(float3),
    v_new : tensor(float3)
):

    # Get thread ID
    thread_id = tid()

    # Load state variables and parameters
    x0 = load(x, thread_id)
    v0 = load(v, thread_id)
    f0 = load(f, thread_id)
    inv_mass = load(w, thread_id)

    # Load external forces
    g = load(gravity, 0)

    # Semi-implicit Euler
    v1 = v0 + (f0 * inv_mass - g * step(inv_mass)) * dt
    x1 = x0 + v1 * dt

    # Store results
    store(x_new, thread_id, x1)
    store(v_new, thread_id, v1)
```

Listing 1: Particle Integration Kernel

At runtime, the kernel's abstract syntax tree (AST) is parsed using Python's built-in `ast` module. We then generate C++ kernel code for forward and reverse mode, which may be compiled to a CPU or GPU executable using the PyTorch `torch.utils.cpp_extension` mechanism.

This approach allows writing imperative code, with fine-grained indexing and implicit operator fusion (since all operations in a kernel execute as one GPU kernel launch). Each kernel is wrapped as a PyTorch autograd operation so that it fits natively into the larger computational graph.

A4 MPC Controller Architecture

For our model predictive control examples, we have used a simple 3-layer neural network architecture illustrated in Figure 7. With simulation time t as input we generate N phase-shifted sinusoidal signals which are passed to a fully-connected layer (zero-bias), and a final activation layer. The output is a vector of per-element activation values as described in the previous section.

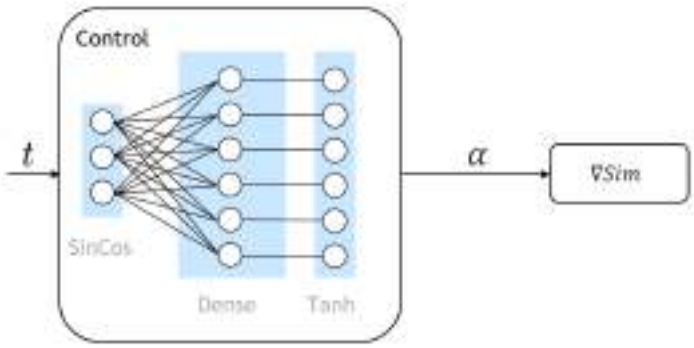


Figure 7: Our simple network architecture used the for `control-walker` and `control-fem` tasks.

A5 Loss landscapes for parameter estimation of deformable solids

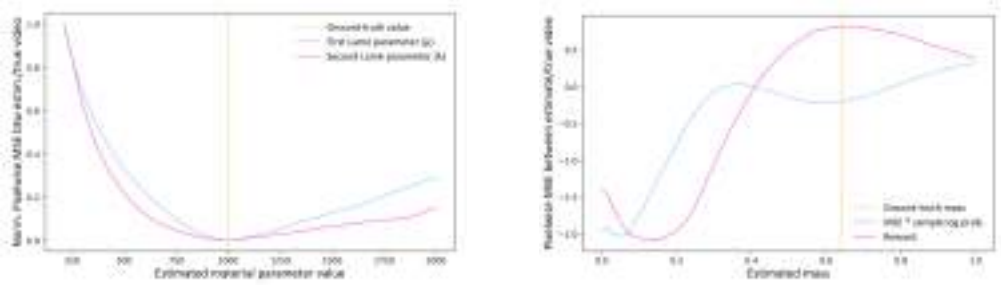
∇Sim integrates several functional blocks, many of which contain nonlinear operations. Furthermore, we employ a pixelwise mean-squared error (MSE) loss function for estimating physical parameters from video. To demonstrate whether the gradients obtained from ∇Sim are relevant for the task of physical parameter estimation, in Figure 2 of the main paper, we present an analysis of the MSE loss landscape for mass estimation.

A5.1 Elasticity parameter

We now present a similar analysis for elasticity parameter estimation in deformable solids. Figure 8a shows the loss landscape when optimizing for the Lamé parameters of a deformable solid FEM. In this case, both parameters λ and μ are set to 1000. As can be seen in the plot, the loss landscape has a unique, dominant minimum at 1000. We believe the well-behaved nature of our loss landscape is a key contributing factor to the precise physical-parameter estimation ability of ∇Sim.

A5.2 Loss landscape in PyBullet (REINFORCE)

Figure 8b shows how optimization using REINFORCE can introduce complications. As the simulation becomes unstable with masses close to zero, poor local optimum can arise near the mean of the current estimated mass. This illustrates that optimization through REINFORCE is only possible after careful tuning of step size, sampling noise and sampling range. This reduces the utility of this method in a realistic setting where these hyperparameters are not known a priori.



(a) **Lamé loss landscape** (b) **PyBullet loss landscape**

Figure 8: **Loss Landscapes**: (left) when optimizing for the elasicity parameters of a deformable FEM solid. Both the Lamé parameters λ and μ are set to 1000, where the MSE loss has a unique, dominant minimum. (right) when optimizing for the mass, the reward (negative normalized MSE) has a maximum close to the ground truth maximum but the negative log likelihood of each mass sample that's multiplied with the reward only shows a local minimum that's sensitive to the center of the current mass estimate.

A6 Dataset details

For the rigid-body task of physical parameter estimation from video, we curated a dataset comprising of 14 meshes, as shown in Fig. 9. The objects include a combination of primitive shapes, fruits and vegetables, animals, office objects, and airplanes. For each of the experiments, we pick an object at random, and sample its physical attributes from a predefined range: densities are in the range $[2, 12]\, kg/m^3$, contact parameters k_e, k_d, k_f are in the range $[1, 500]$, and the coefficient of friction μ is in the range $[0.2, 1.0]$. The positions, orientations, (anisotropic) scale factors, and initial velocities are randomly uniformly sampled in a cube of side-length $13m$ centered at the camera. Across all rigid-body experiments, we use 800 objects for training and 200 objects for testing.

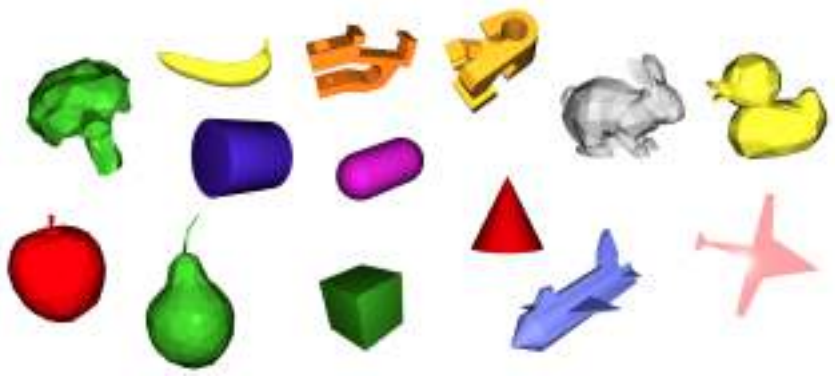


Figure 9: **Objects used** in our rigid-body experiments. All of these meshes have been simplified to contain 250 vertices or fewer, for faster collision detection times.

A7 Baselines

In this section, we present implementation details of the baselines we used in our experiments.

A7.1 ∇PyBullet: PyBullet + REINFORCE

To explore whether existing non-differentiable simulators can be employed for physical parameter estimation, we take PyBullet [17] – a popular physics engine – and make it trivially differentiable, by gradient estimation. We employ the REINFORCE [18] technique to acquire an approximate gradient through the otherwise non-differentiable environment. The implementation was inspired by [19] and [20]. In concurrent work, a similar idea was explored in [21].

Parameter	Value	Meaning
no_samples	5	How often was the mass sampled at every step
optimization_steps	125	Total number of optimization steps
sample_noise	0.05	Std. dev. of normal distribution that mass is sampled from
decay_factor	0.925	Factor that reward decay is multiplied with after optimizer step
dataset_size	200	Number of bodies that the method was evaluated on

Table 4: PyBullet-REINFORCE hyperparameters.

In PyBullet, the mass parameter of the object is randomly initialized in the range $[0, N_v]$, where N_v is the number of vertices, the object is set to the same starting position and orientation as in the dataset, and the camera parameters are identical to those used in the dataset. This configuration ensures that if the mass was correct, the video frames rendered out by PyBullet would perfectly align with those generated by ∇*Sim*. Each episode is rolled out for the same duration as in the dataset (60 frames, corresponding to 2 seconds of motion). In PyBullet this is achieved by running the simulation at 240 Hz and skipping 7 frames between observations. The REINFORCE reward is calculated by summing the individual $L2$ losses between ground truth frames and PyBullet frames, then multiplying each by -1 to establish a global maximum at the correct mass, as opposed to a global minimum as in ∇*Sim*. When all individual frame rewards have been calculated, all trajectory rewards are normalized before calculating the loss. This ensures that the reward is scaled correctly with respect to REINFORCE's negative sample log likelihood, but when the mass value approaches the local optimum, this leads to instability in the optimization process. To mitigate this instability, we introduce reward decay, which a hyperparameter that slowly decreases the reward values as optimization progresses, in a similar manner to learning rate decay. Before each optimization step, all normalized frame reward values are multiplied by $reward\_decay$. After the optimization step, the decay is updated by $reward\_decay = reward\_decay * decay\_factor$. The hyperparameters used in this baseline can be found in Table 4.

A7.2 CNN for direct parameter estimation

In the rigid-body parameter estimation experiments, we train a convnet baseline, building on the EfficientNet-B0 architecture [22]. The convnet consists of 2 convolutional layers with parameters (PyTorch convention): $(1280, 128, 1)$, $(128, 32, 1)$, followed by linear layers and Relu activations with sizes $[7680, 1024, 100, 100, 100, 5]$. No activation is applied over the output of the convnet. We train the model to minimize the mean-squared error between the estimated and the true parameters, and use the Adam optimizer [23] with learning rate of 0.0001. Each model was trained for 100 epochs on a $V100$ GPU. The input image frames were preprocessed by resizing them to 64×64 pixels (to reduce GPU memory consumption) and the features were extracted with a pretrained EfficientNet-B0.

A8 Compute and timing details

Most of the models presented in ∇*Sim* can be trained and evaluated on modern laptops equipped with graphics processing units (GPUs). We find that, on a laptop with an i7 processor and a GeForce GTX 1060 GPU, parameter estimation experiments for rigid/nonrigid bodies can be run in about 5-20 minutes per object, and the visuomotor control experiments (`control-fem`, `control-cloth`) take about 30 minutes per episode. We are actively working on code/resource optimization to improve runtime performance and overall system resource usage, across the physics and rendering engines.

A9 Overview of available differentiable simulations

Table 5 presents an overview of the differentiable simulations implemented in ∇*Sim*, and the optimizable parameters therein.

	pos	vel	mass	rot	rest	stiff	damp	actuation	g	μ	e	ext forces
Rigid body	✓	✓	✓	✓					✓	✓	✓	
Simple pendulum	✓								✓			✓
Double pendulum	✓								✓			✓
Deformable object	✓	✓	✓	✓				✓	✓	✓	✓	
Cloth	✓	✓	✓		✓	✓	✓			✓		
Fluid (Smoke) (2D)		✓										

Table 5: An overview of **optimizable parameters** in ∇Sim. Table columns are (in order, from left to right): Initial particle positions (pos), Initial particle velocities (vel), Per-particle mass (mass), Initial object orientation (rot), Spring rest lengths (rest), Spring stiffnesses (stiff), Spring damping coefficients (damp), Actuation parameters (actuation), Gravity (g), Friction parameters μ, Elasticity parameters (e), External force parameters (ext forces).

A10 Limitations

While providing a wide range of previously inaccessible capabilities, ∇Sim has a few limitations that we discuss in this section. These shortcomings also form interesting avenues for subsequent research.

- ∇Sim (and equivalently ∇PyBullet) are inept at handling **tiny masses** ($100g$ and less). Optimizing for physical parameters for such objects requires a closer look at the design of physics engine and possibly, numerical stability.
- **Articulated bodies** are not currently implemented in ∇Sim. Typically, articulated bodies are composed of multiple prismatic joints which lend additional degrees of freedom to the system.
- While contacts with simple geometries (such as between arbitrary triangle meshes and planar surfaces) are handled, ∇Sim has limited capability to handle **contact-rich** motion that introduces a large number of discontinuities. One way to handle contacts differentiably could be to employ more sophisticated contact detection techniques and solve a *linear complementarity problem* (LCP) at each step, as done in [7].
- Aside from the aforementioned drawbacks, we note that physics engines are adept at modeling phenomena which can be codified. However, there are several **unmodeled physical phenomena** that occur in **real-world** videos and must be tackled in order for ∇Sim to evolve as a scalable framework capable of operating in the wild.

References

[1] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. Springer Science & Business Media, 2006. 1

[2] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE, 2015. 1, 2

[3] Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Transactions on Graphics (TOG)*, 37(2):1–15, 2018. 2

[4] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *International Conference on Learning Representations*, 2020. 2, 3

[5] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *Proceedings of International Conference on Computer Vision*, 2019. 2

[6] Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Neural Information Processing Systems*, 2019. 2

[7] 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 31*, pages 7178–7189. Curran Associates, Inc., 2018. 2, 3, 7

[8] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis Wyffels. A differentiable physics engine for deep learning in robotics. *CoRR*, abs/1611.01652, 2016. 3

[9] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks, 2020. 3

[10] Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks, 2019. 3

[11] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks, 2019. 3

[12] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators, 2019. 3

[13] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999. 3

[14] Matt Johnson Dougal Maclaurin, David Duvenaud and Jamie Townsend. Autograd, 2015. 3

[15] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019. 3

[16] Charles C Margossian. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(4):e1305, 2019. 3

[17] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019. 5

[18] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, May 1992. 5

[19] Jiajun Wu, Ilker Yildirim, Joseph J Lim, William T Freeman, and Joshua B Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Neural Information Processing Systems*, 2015. 5

[20] Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. In *Advances in neural information processing systems*, pages 4996–5004, 2016. 5

[21] Kiana Ehsani, Shubham Tulsiani, Saurabh Gupta, Ali Farhadi, and Abhinav Gupta. Use the force, luke! learning to predict physical forces by simulating effects. In *CVPR*, 2020. 5

[22] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019. 6

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 6