IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 16 May 2024, accepted 4 July 2024, date of publication 9 July 2024, date of current version 23 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3425448

SURVEY

A Review of Differentiable Simulators

RHYS NEWBURY[1,2], JACK COLLINS[3], (Member, IEEE), KERRY HE[1], JIAHE PAN[4], INGMAR POSNER[3], (Member, IEEE), DAVID HOWARD[5], (Member, IEEE), AND AKANSEL COSGUN[6]

[1]Department of Electrical and Computer System Engineering, Monash University, Clayton, VIC 3800, Australia
[2]College of Engineering and Computer Science, The Australian National University, Canberra, ACT 2601, Australia
[3]Applied AI Lab, Oxford Robotics Institute, University of Oxford, OX1 2JD Oxford, U.K.
[4]Faculty of Engineering and Information Technology, The University of Melbourne, Melbourne, VIC 3010, Australia
[5]CSIRO, Brisbane, QLD 4069, Australia
[6]School of Information Technology, Deakin University, Melbourne, VIC 3125, Australia

Corresponding author: Rhys Newbury (rhys.newbury@monash.edu)

This work was supported by the UKRI/EPSRC Program under Grant EP/V000748/1.

ABSTRACT Differentiable simulators continue to push the state of the art across a range of domains including computational physics, robotics, and machine learning. Their main value is the ability to compute gradients of physical processes, which allows differentiable simulators to be readily integrated into commonly employed gradient-based optimization schemes. To achieve this, a number of design decisions need to be considered representing trade-offs in versatility, computational speed, and accuracy of the gradients obtained. This paper presents an in-depth review of the evolving landscape of differentiable physics simulators. We introduce the foundations and core components of differentiable simulators alongside common design choices. This is followed by a practical guide and overview of open-source differentiable simulators that have been used across past research. Finally, we review and contextualize prominent applications of differentiable simulation. By offering a comprehensive review of the current state-of-the-art in differentiable simulation, this work aims to serve as a resource for researchers and practitioners looking to understand and integrate differentiable physics within their research. We conclude by highlighting current limitations as well as providing insights into future directions for the field.

INDEX TERMS Differentiable simulator, review, differentiable physics, soft body simulation, system identification, trajectory optimization, morphology optimization, policy optimization, robotics.

I. INTRODUCTION

Physics simulators are extensively utilized within the sciences, and are a key enabling technology for a range of industrial, design, engineering, and robotics applications [1]. These simulators are grounded in mathematical models of physical laws, coupled to pertinent constraints, *e.g.* joint and velocity limits for robotics applications. This allows for a principled approach to numerically predict forward in time given the current system state.

Simulators have increasingly leveraged rapidly evolving hardware resources (*i.e.* multi-threaded CPUs and GPUs, high performance compute clusters) to execute parallel simulations far faster than real-time, allowing for large amounts of data to be collected simultaneously. This has made simulation a key tool for machine learning applications, particularly in scenarios where data is not readily available, such as robotics [2], [3].

Despite providing computationally-tractable data generation, traditional physics simulators critically do not provide access to the gradient information that machine learning heavily relies on to drive the learning process. This limitation has spurred the development of a new class of *differentiable* simulators (see Fig. 1 for a visualization of the breadth of published differentiable simulator research). Differentiable simulators are end-to-end differentiable, *i.e.* capable of calculating gradients throughout the entire duration of a simulation for a given loss function with respect to desired parameters (*e.g.* surface friction with respect to a mean squared error loss comparing the simulated trajectory and a ground-truth trajectory). Thus, they provide a direct route for integration within gradient-based optimization frameworks

The associate editor coordinating the review of this manuscript and approving it for publication was Zheng H. Zhu.

and deep learning pipelines. Since the initial exploration of a general-purpose differentiable simulator by Degrave et al. [4], the field has grown rapidly, to the point where there are several well-supported and maintained differentiable physics simulators.

Recognizing the emergence of this field of research, this review aims to provide a timely and comprehensive overview of differentiable simulators – what they are, how they work, and how have they been used in research thus far. Additionally, we curate an up-to-date list of currently supported and well-maintained differentiable physics simulators, serving as a reference for practitioners when selecting suitable options for their specific applications. To ensure comprehensive coverage, this review encompasses all relevant literature published before 2024 that proposes or utilizes differentiable physics simulators, with some necessary caveats which we will outline in the relevant sections. In summary, the contributions of this paper are, (i) a thorough review of the fundamental concepts and methodologies of differentiable physics simulators, (ii) a curated list of well-maintained differentiable simulators to aid practitioners in selecting appropriate tools, (iii) applications of differentiable simulators demonstrating how these simulators have been used in the past, and (iii) a discussion on future directions of the field.

The review comprises the following sections: Foundations of Differentiable Simulators (Section II), which discusses the core components of a differentiable physics simulator and what makes them differentiable; Differentiable Simulators (Section III), covering a subset of open-source differentiable simulators utilized in past research, and therefore this selection offers a good starting point for exploring the practical side of differentiable simulators; Applications (Section IV) including system identification, trajectory optimization, policy optimization and morphology optimization; and, Discussion (Section V) with a future outlook of the field of differentiable simulators.

II. FOUNDATIONS OF DIFFERENTIABLE SIMULATORS

A differentiable simulator is comprised of several core components.

1) Gradient Calculation – this is the component of differentiable simulators which distinguishes them from standard physics simulators, and is responsible for computing gradients with respect to the simulator's parameters.
2) Dynamics Model – the underlying physics governing the simulated system. For the purposes of this review, we restrict our attention to simulators whose physics are governed by a predefined set of equations based on physical kinematic or dynamic constraints.
3) Contact Model – most differentiable simulators implement a contact model to simulate interactions between objects during collision events. These contact models introduce a unique challenge to calculating gradients due to the inherently discontinuous nature of contacts.
4) Integrator – an integrator numerically solves the equations of motion over discrete time steps. Although the derivation and computation of gradients through explicit integration schemes is fairly straightforward, implicit integration schemes, which usually require solving a nonlinear system of equations to obtain the state at the next time step, introduce unique challenges.

These components collectively form the underlying substrate of a differentiable physics simulator. Fig. 2 presents a simplified visualization of how the different components interact to produce gradients of a physics simulation that can be used for optimization.

The scope of this review is differentiable physics simulators with the exclusion of neural networks trained on simulated data, *e.g.* [16], [17], [18], [19], [20], [21], [22], [23], [24], as although they provide a method of deriving gradients that are physics informed, they are not constrained to conform to the underlying physics model. Additionally, we exclude traditional simulators employing finite differencing for derivative calculations, such as [25], as purpose-built differentiable simulators offer derivatives with speed and accuracy not achievable by traditional simulators. Excluding such avenues for calculating physics-informed gradients ensures a more focused examination of the techniques used when developing the main identified components of differentiable simulators, without diluting the primary points of interest. However, we acknowledge that this reduced scope of our review will exclude some seminal work in the field of differentiable simulation.

Throughout this section, $q, \dot{q}, \ddot{q} \in \mathbb{R}^n$ represent the n-dimensional generalized coordinates of the system, their time derivatives, and second time derivates (accelerations), respectively. Many sources do not explicitly state parts of their methodology, so where possible the code was examined to try and ascertain the correct information. Papers where the required information could not be sourced are not included in the following subsections. An overview of all the differentiable physics engines covered in this review coupled with the high-level design choices made for each is included in Table 1.

A. GRADIENT CALCULATION

There are many methods to calculate gradients. In the following, we will review automatic differentiation in Section II-A1, symbolic differentiation in Section II-A2, and analytic derivations of gradients in Section II-A3.

The majority of simulation frameworks reviewed in this paper use automatic differentiation as the backbone for their gradient computations. This is because automatic differentiation avoids the need to manually derive gradients by hand, which can be increasingly cumbersome the more complex the simulator is. In comparison, symbolic differentiation, despite also automating the evaluation of the gradient of a given function, needs to build symbolic expressions of the gradient which grow exponentially with the size of the function being differentiated [66], and is therefore

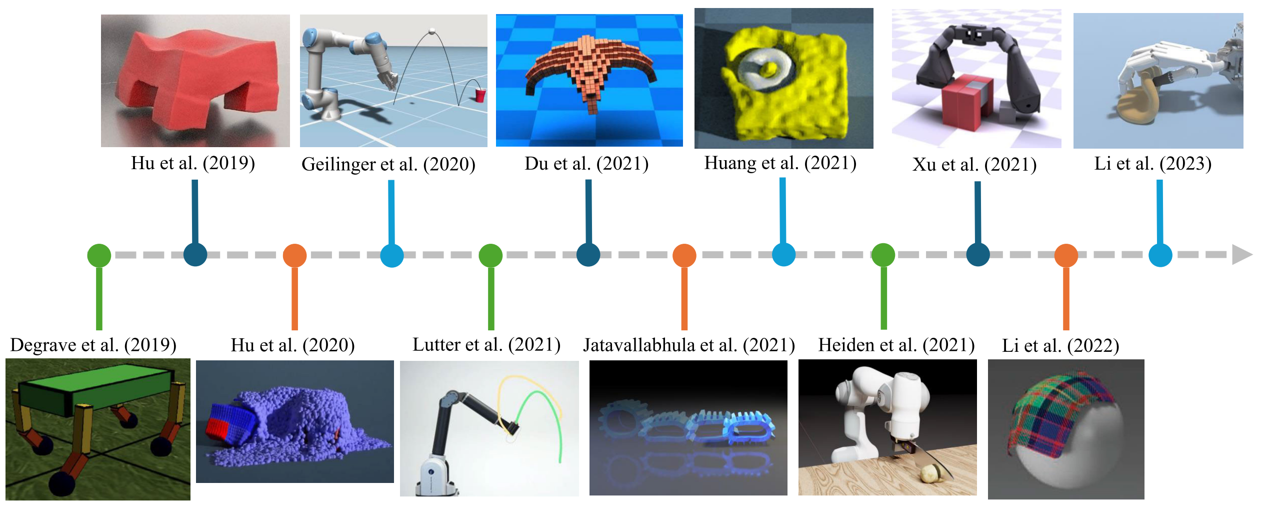


FIGURE 1. A visualization of the breadth of published research progressing the field of differentiable simulation. Areas and applications of differentiable simulators cover topics such as soft and rigid-body simulation; system identification; trajectory optimization; morphology optimization; and many others covered in-detail in this review [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] (© 2024 IEEE).

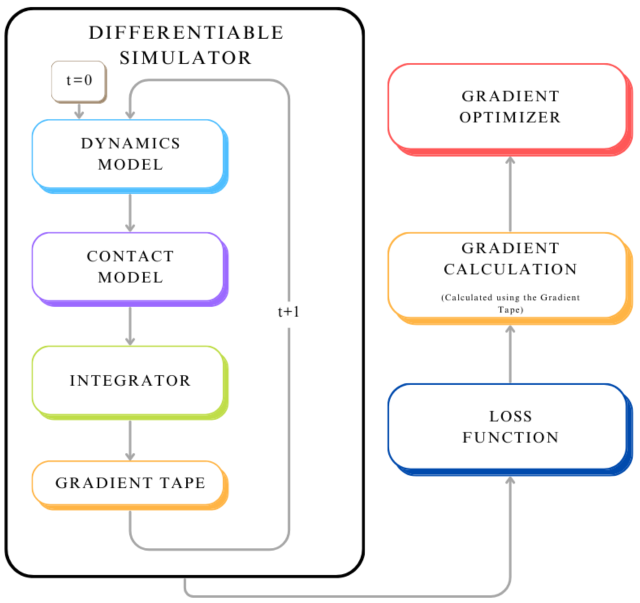


FIGURE 2. An overview of how the different components of a differentiable simulator interact. Each of these components (except for the loss function and gradient optimizer) are explored in detail in Section II.

rarely used over automatic differentiation in differentiable simulators. Both of these automated gradient calculation tools incur an overhead which can be disadvantageous in performance critical applications. Similarly, it can be difficult to apply these methods on dynamics which are discontinuous or given as, *e.g.*, a solution of a feasibility problem rather than an explicit function. Both of these issues arise when modelling contacts (see Section II-C). In these scenarios, it can be advantageous to analytically derive expressions for the gradients instead, or to use analytical techniques (*e.g.*, from linear algebra) to simplify the computation of the gradient. Overall, we are aware of only one work which does not use automatic differentiation [32], instead calculating all gradients analytically.

1) AUTOMATIC DIFFERENTIATION

Automatic differentiation (AD) relies on the fact that even complex functions are composed of elementary operations (*e.g.*, addition, subtraction, etc.) and functions (*e.g.*, sin, cos, exp, etc.), which allows for repeatedly applying the chain rule to these expressions in a programmatic way to compute the value of a derivative. There are two main types of AD: forward mode and reverse mode. Forward mode AD traverses through the chain rule starting from the function input, and is more efficient when there are more outputs than inputs. In comparison, reverse mode AD starts from the function output, and is more efficient in cases where there are more inputs than outputs [66]. As differentiable simulators are typically used to find the derivative of a (scalar valued) objective function with respect to many simulation parameters, reverse mode AD is typically used for these applications. An alternative approach to AD is source code transformation, which is used by Hu et al. [5]. This uses a just-in-time (JIT) transpiler over Python code to produce a function which will calculate gradients.

Many differentiable simulators [4], [8], [9], [15], [26], [27], [28], [30], [33], [34], [36], [40], [41], [43], [45], [47], [49], [51], [52], [53], [57], [59], [60], [61], [64] use a pipeline based *solely* on AD, commonly using either PyTorch Autograd [67], JAX [68] or DiffTaichi [5].

2) SYMBOLIC DIFFERENTIATION

Like AD, symbolic differentiation computes the derivative of a function by applying the chain rule repeatedly. However, whereas AD computes a numerical value for the derivative, symbolic differentiation produces a symbolic expression. However, for applications using differentiable simulators,

TABLE 1. A table of all the differentiable physics simulators found throughout our review. This does not include every paper within the scope of the review, only those papers which present a new engine for differentiable simulation, not application-based papers which use existing engines. We highlight important components of differentiable simulators as columns within the table. For more details on each column, see Section II-A for Gradient Methods, Section II-B for Dynamical Models, Section II-C for Contact Models, and Section II-D for Integrators.

Paper	Gradient Method	Dynamical Model	Soft Body	Contact Model	Integrator
de Avila Belbute-Peres et al. (2018)	Auto Diff	Newton	-	LCP	Explicit
Degrave et al. (2019)	Auto Diff	Newton	-	Complementarity	Semi-Implicit
Heiden et al. (2019) **(TDS)**	Auto Diff	Newton	-	Compliant Model	Semi-Implicit
Heiden et al. (2019)	Auto Diff	Newton	-	None	Semi-Implicit
Hu et al. (2019)	Analytical Symbolic	Continuum Mechanics	✔	MLS-MPM	Explicit
Liang et al. (2019)	Implicit Diff	Newton	✔	Position Based	Implicit
Geilinger et al. (2020)	Adjoint Method	Newton	✔	NCP	Implicit
Holl et al. (2020)	Adjoint Method	Fluid Simulation	-	None	Explicit
Holl et al. (2020) **(PhiFlow)**	Auto Diff	Fluid Simulation	-	Explicit	None
Qiao et al. (2020)	Implicit Diff	Newton	-	Position Based	Implicit
Song and Boularias (2020)	Analytical	Newton	-	None	Explicit
Ding et al. (2021)	Auto Diff	Newton	-	Impulse Based	Implicit
Du et al. (2021)	Adjoint Method	Projective Dynamics	✔	Complementarity	Implicit
Du et al. (2021)	Auto Diff	Projective Dynamics	✔	Complementarity	Implicit
Freeman et al. (2021) **(Brax)**	Newton	XPBD	-	Position Based	Semi-Implicit
Heiden et al. (2021)	Auto Diff	Newton	-	NCP + Compliant	Semi-Implicit
Huang et al. (2021)	Auto Diff	Continuum Mechanics	✔	MLS-MPM	Explicit
Jatavallabhula et al. (2021) **(GradSim)**	Auto Diff	Newton	✔	Compliant Model	Semi-Implicit
Le Lidec et al. (2021)	Implicit Diff	Lagrangian	-	Complementarity	Implicit
Lutter et al. (2021)	Auto Diff	Newton	-	None	Not Specified
Mora et al. (2021)	Adjoint Method	Newton	-	None	Implicit
Qiao et al. (2021)	Adjoint Method	Newton	-	LCP	Explicit
Ścibior et al. (2021)	Auto Diff	Newton	-	None	Not Specified
Wang et al. (2021)	Auto Diff	Newton	-	Impulse Based	Semi-Implicit
Werling et al. (2021) **(Nimble)**	Symbolic	Lagrangian	-	LCP	Explicit
Xu et al. (2021)	Auto Diff	Newton	-	Compliant Model	Semi-Implicit
Zhong et al. (2021)	Implicit Diff	Lagrangian	-	Convex Optimization	Explicit
Gärtner et al. (2022)	Auto Diff	Newton	-	LCP	Semi-Implicit
Gong et al. (2022)	Implicit Diff	Lagrangian	✔	Compliant Model	Implicit
Granados et al. (2022)	Auto Diff	Newton	-	None	Explicit
Howell et al. (2022) **(Dojo)**	Implicit Diff	Lagrangian		Complementarity	Implicit
Nava et al. (2022)	Auto Diff	Fluid Sim Continuum Mechanics	✔	None	Implicit
Petrík et al. (2022)	Auto Diff	Newton	-	Impulse Based	Explicit
Turpin et al. (2022)	Auto Diff	Newton	-	Compliant Model	Semi-Implicit
Wang et al. (2022)	Auto Diff	Newton	-	None	Implicit
Wang et al. (2022)	Auto Diff	Newton	-	Impulse Based	Implicit Semi-Implicit
Zhao et al. (2022)	Auto Diff	Newton Fluid Simulation	-	None	Explicit
Zhao et al. (2022)	Analytical	Newton Fluid Simulation	-	None	Not Specified
Stuyck and Chen (2023)	Analytical	XPBD	✔	Implicit	Compliant Model
Bezgin et al. (2023) **(JAX-Flows)**	Auto Diff	Fluid Simulation		Explicit	None
Chen et al. (2023) **(daX)**	Auto Diff	Continuum Mechanics	✔	MLS-MPM	Explicit
Le Cleac'h et al. (2023)	Auto Diff	Newton	-	Compliant Model	Implicit
Liu et al. (2023)	Auto Diff	XPDB	✔	Position Based	Explicit
Spielberg et al. (2023)	Auto Diff	Continuum Mechanics	✔	MLS-MPM	Explicit
Turpin et al. (2023)	Auto Diff	Newton	-	Position Based	Semi-Implicit
Wang et al. (2023)	Auto Diff	Continuum Mechanics	✔	MLS-MPM	Semi-Implicit
Wiedemann et al. (2023)	Auto Diff	Newton	-	None	Explicit
Xian et al. (2023) **(FluidLab)**	Analytical	Continuum Mechanics Fluid Simulation		MLS-MPM	Explicit

usually we are only interested in the gradient evaluated at a single point to perform gradient based optimization, and therefore the full symbolic gradient is not required. Additionally, as previously discussed, the main drawback of symbolic differentiation is that the size of the symbolic gradients grow rapidly with the complexity of the function. Therefore, symbolic differentiation is typically only used when the dynamics are simple, or for a few select components of the simulator. This is done in [6], [14], and [42].

3) ANALYTICAL GRADIENTS

It can be challenging to apply AD and symbolic differentiation to discontinuous processes or phenomena. In these situations, it is advantageous to *manually* derive explicit expressions for derivatives of functions arising from differentiable simulators. Also, while AD and symbolic differentiation require the computation of gradients online, analytical gradients provide an explicit expression for the gradient a priori, allowing gradients to be computed more quickly and accurately. Another situation where it can be difficult to apply AD and symbolic differentiation is when expressions are given as a feasibility problem, which often arises in contact modeling (see Section II-C). Such components often also require analytic derivations of gradients.

The gradients of an entire simulator can be derived manually. For example, [32] derive analytical expressions for an environment with a sliding object. However, this can be very cumbersome for more complex scenarios, which involve many bodies with complex contact dynamics. In comparison, [14], [55] only manually derive gradients for performance critical components of their system.

When manually deriving the explicit expressions for gradients, authors often follow standard techniques such as implicit differentiation or the adjoint method. These methods will be discussed in the remainder of this subsection.

Implicit Differentiation: Implicit differentiation is a technique used to obtain a derivative dy/dx from an expression $f(y, x) = 0$ containing the two variables, even when one variable is not explicitly represented as a function of the other, *i.e.*, as $y(x)$. This technique can be particularly useful for contact models which are expressed as feasibility or optimization problems, which can be difficult to obtain gradients for using AD or symbolic differentiation.

Liang et al. [29] and Gong et al. [46] use implicit differentiation to derive the gradients for a Newtonian and Lagrangian dynamic model, respectively. Howell et al. [48] and Le Cleac'h et al. [59] use the implicit function theorem on a set of optimality conditions to compute the gradients of a contact model given as the solution of a feasibility problem.

In OptNet [69], it was shown how implicit differentiation could be used to obtain gradients through an expression given by a Quadratic Program (QP), which are optimization problems of the form

$$\min_z \quad \frac{1}{2}z^T Qz + q^T z \tag{1a}$$

$$\text{subj. to} \quad Az = b \tag{1b}$$

$$Gz \leq h. \tag{1c}$$

Subsequent research has extended this formulation to address diverse constraints. Liang et al. [29] refine the implicit differentiation technique of a QP to minimize the resulting size of the linear system. This is achieved using QR decomposition, a well-known matrix factorization method. Expanding on this, Qiao et al. [31] further extends the methodology to handle nonlinear constraints by incorporating information from the Jacobian. Additionally, Le Lidec et al. [37] extends these methods to address quadratically constrained quadratic programs (QCQPs), *i.e.*, problems of the form Eq. (1) but with the linear inequality constraint Eq. (1c) replaced with p quadratic constraint $z^\top G_i z + g_i^\top z \leq h_i$ for $i = 1, \ldots, p$. Furthermore, the gradients for a linear complementarity problem (LCP) formulation (see Section II-C1 for more detail) can also be derived using similar techniques [26].

Adjoint Method: The adjoint method leverages an adjoint vector obtained by solving a linear system to make the process of obtaining gradients less computationally expensive. This adjoint vector encapsulates information about how changes in the system's output (*e.g.*, the final state) relate to changes in the parameters. Using this adjoint vector in the gradient calculation allows efficient calculation of parameters that influence the system behavior without directly computing how each parameter affects the entire trajectory.

More concretely, consider the derivative of a loss function $\mathcal{L}$ with respect to some parameters θ

$$\frac{d\mathcal{L}}{d\theta} = \frac{\partial \mathcal{L}}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial q}^\top \frac{dq}{d\theta}. \tag{2}$$

The main challenge is in computing $\frac{dq}{d\theta}$, which can be obtained by implicitly differentiating through the system dynamics $f(q, \theta) = 0$. However this is expensive as it involves solving a large number of linear equations. The adjoint method avoids this by first computing the adjoint vector $z = (\frac{\partial f}{\partial q})^{-\top} \frac{\partial \mathcal{L}}{\partial q}$ by solving a single linear system, then computes the desired gradient as

$$\frac{d\mathcal{L}}{d\theta} = \frac{\partial \mathcal{L}}{\partial \theta} - z^\top \frac{\partial f}{\partial \theta}. \tag{3}$$

This method is adopted by [6], [7], [11], [13], [39], and [56] for computations of gradients of the loss function. In [38], an extension of the adjoint method by [70] was used to obtain the Hessian of the loss function, which can be used to accelerate the convergence of optimization algorithms.

B. DYNAMICS MODEL

A dynamics model is a mathematical model that describes the behavior of a system over time. It can be used to predict how the system will respond given changes to its inputs. There are distinct approaches when modeling dynamic systems, with three prominent categories: rigid body dynamics, soft body dynamics and fluid dynamics. Rigid body and soft body dynamics are strongly linked. However, a key distinction is that soft body dynamics incorporates internal forces within the dynamics model. This inclusion enables a more faithful representation of the complex interactions and deformation characteristics of soft materials. In rigid body dynamics, Newtonian or Lagrangian methods are typically used for modeling the system's dynamics. However, these methods can be extended to handle soft bodies with the inclusion of internal forces. There are other dynamics models which are purpose-built for modeling soft body objects, such as continuum mechanics [14], Projective Dynamics [71], or Compliant Position-Based Dynamics (XPBD) [72]. Fluid dynamics, distinct from rigid and soft body dynamics, involves the study of fluid flow patterns and interactions within fluidic environments.

1) RIGID BODY DYNAMICS

The **Newtonian** dynamics model is based on Newton's laws of motion, which expresses the acceleration of a body as a function of the forces acting on it and its mass, *i.e.*,

$$F = m\ddot{q}, \tag{4}$$

where F is the force acting on the body, m is the mass of the body, and $\ddot{q}$ is the acceleration of the body. However, this neglects the rotational motion of the body and only considers the translational motion. Newton-Euler takes into account the rotational motion of the body, *i.e.*,

$$M(q)\ddot{q} + C(q, \dot{q}) = \tau, \tag{5}$$

where τ is a vector of generalized forces, M is the mass matrix, and C is the bias force matrix which accounts for other forces acting on the system, including centrifugal forces, Coriolis forces and gravity.

Several approaches use Newton or Newton-Euler dynamics to model their rigid-body system [4], [6], [15], [26], [27], [28], [29], [31], [32], [35], [38], [41], [43], [45], [52], [53], [54], [55], [59], [62], [64]. Granados et al. [47] apply Newton's method but only consider the steady-state response arising from the model.

Many works incorporate kinematic constraints into Newtonian dynamics for their specific problem domain. For example, Turpin et al. [51] use a slight extension to Newtonian dynamics to account for the kinematics of a robotic hand. Heiden et al. [36], Qiao et al. [39] uses the Articulated Body Algorithm [73] which is based on Newton-Euler recursive dynamics. Ścibior et al. [40] adopt a simple differentiable model, a kinematic bicycle.

Some works frame Newtonian dynamics as an impulse-based model rather than a force-based model. Ding et al. [33], Petrík et al. [50] make use of an impulse-based model, using conservation of momentum to calculate the change in velocities when there are collisions of objects. Jatavallabhula et al. [8] updates momentum in terms of $F = \frac{dp}{dt}$, where p is the linear momentum of an object. An integration step is then performed to calculate the new momentum and the updated linear velocity can be computed as $v = p/m$.

The **Lagrangian** dynamics formulation is based on the principle of least action, which states that the motion of a system will minimize the action, which is a function of the system's configuration and its time derivatives.

The Lagrangian for a simple particle is given by

$$\mathcal{L} = T - V, \tag{6}$$

where T is the kinetic energy of the particle, and V is the potential energy of the particle. The Euler-Lagrange equation, which describes the equations of motion for a system in terms of its Lagrangian, is given by

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} = 0. \tag{7}$$

Lagrangian dynamics are adopted by [37], [42], [46], [74], and [75]. In comparison, Howell et al. [48] derive their physics by using Hamilton's Principle of Least-Action. By directly performing integration on these equations, the simulator is able to automatically conserve momentum and energy in the system. It is important to note that Newton, Lagrangian, and Hamiltonian dynamics are all equivalent models [76] but provide different ways to formulate the equations.

Extending Rigid Body to Soft Body: To extend rigid body dynamics to soft bodies, additional considerations need to be included for internal forces within the soft body. These internal forces address how the soft body resists changes in shape, volume, and overall configuration. Internal forces have been used in both Newton dynamical model [6], [8], [29] and Lagrangian models [46]. A Neo-Hookean material model is used in [6], [8] which describes the energy density as a function of the deformation gradient. An FEM method is adopted by [29], where for each triangle face in the mesh, the deformation gradient is calculated as a variable of the strain. Gong et al. [46] model cloth as woven elastic rods, and derive the equations of force from [77].

2) SOFT BODY

Continuum Mechanics deals with the mechanical behavior of materials modeled as continuous substances, rather than as collections of discrete particles. It provides a framework for analyzing the motion and deformation of solids and fluids. The governing equations of continuum mechanics are typically expressed in terms of the conservation of mass and momentum [78]. Momentum conservation, also known as Newton's Second Law for a Continuous Material, states that the change in momentum of a material element is equal to the sum of the divergence of the stress tensor and the gravitational forces acting on the element, *i.e.*,

$$\rho \frac{Dv}{Dt} = \nabla \cdot \sigma + \rho g, \tag{8}$$

where ρ represents the material density, v is the velocity of the particle, σ is the Cauchy stress tensor and $\frac{D\phi}{Dt} \equiv \frac{\partial \phi}{\partial t} + v\dot{\nabla}(q)$ is the material derivative of any function $\phi(x, t)$. Mass conservation states that the rate of change of density plus the divergence of the mass flux (density multiplied by velocity) is zero, *i.e.*,

$$\frac{D\rho}{Dt} + \rho \nabla \cdot v = 0. \tag{9}$$

Continuum mechanics is adopted by MLS-MPM models [9], [14], [58], [61], [63]. MLS-MPM [78] computes the deformation gradient by reusing quantities computed from the Affine Particle-In-Cell Method [79]. Simpler models for internal forces have also been adapted by [58] who use a mass-spring system to model a cloth. The MLS-MPM model is extended by [9] to include a von Mises yield criterion for modeling plasticity [80]. According to the von Mises yield criterion, a plasticine particle deforms permanently if the stress exceeds a certain limit, and a correction is needed to account for the material's changed initial state. Reference [10] use a different model for continuum mechanics, using the Finite Element Method (FEM) to compute elastic forces based on a Neo-Hookean constitutive model. This model considers material properties such as the Young's modulus and Poisson's ratio, while aiming to preserve volume during large deformations.

Projective Dynamics is an implicit integration method (see Section II-D2) used in computer graphics which simulates the dynamics of deformable objects [71]. Projective Dynamics models object deformation as a projection onto its rest state, then minimizes an energy function that accounts for internal and external forces to obtain the desired dynamics.

Project Dynamics is extended by DiffPD [7] to be differentiable and then integrated into a differentiable simulator. In the context of differentiable simulators, DiffPD provides a faster method for performing implicit integration while requiring less computational memory. This model is then adapted into other work such as [13] and [34].

Li et al. [13] define the internal forces as $f_{int} = -\nabla E$, where E is the potential energy function, this is based on the original definition proposed by [71]. In contrast to this, a co-rotated linear material model [81] is used by [7] and [34]. This model employs a strain measure derived from a polar decomposition and features an energy function that includes terms accounting for both linear and nonlinear characteristics in the material's response to deformation.

Compliant Position-Based Dynamics (XPBD) [72] is a method for simulating the dynamic behavior of deformable objects. XPBD uses a position-based approach [82], directly computing particle positions instead of calculating velocities and subsequently updating positions. These positions are solved iteratively to satisfy given internal (*e.g.*, distances between particles on the same body) and external constraints (*e.g.*, rigid contacts, external forces). The internal constraints are an alternative approach to calculating internal forces, by constraining the particles according to physical laws, realistic deformations and interactions can be ensured. For example, Liu et al. [60] defines a series of internal constraints for a rope (Shear and Stretch, Bend and Twist, Distance) to ensure realism for the simulated rope.

XPBD is adopted by two differentiable simulation engines, Warp [83] and Brax [35] as well as [60]. However, Brax only uses XPBD for rigid body dynamics, not using any internal constraints. These simulators make use of automatic differentiation libraries to calculate the gradient of the dynamics. Recently, Stuyck and Chen [56] show how gradients of the XPBD framework can be derived analytically.

3) FLUID SIMULATION

Some works also aim to bring differentiable capabilities to the domain of fluid simulation. To model hydrodynamics, [34], [54], [55] use aerodynamic force equations for both drag and lift. They discretize geometry to a triangular mesh and compute the lift and drag forces on each surface triangle. To model fluid flow, [30], [49] design differentiable simulation engines which aim to solve the incompressible Navier-Stokes equations [84]. Um et al. [85] studies advection-diffusion models which describe the temporal evolution of velocity in a fluid system, influenced by advection, diffusion, and external forces, as well as being subject to additional equality constraints. Bezgin et al. [57] aim to design a differentiable framework for computational fluid dynamics (CFD) which allows for the simulation of complex phenomena such as three-dimensional turbulence, compressibility effects, and two-phase flow.

C. CONTACT MODEL

One of the main difficulties in implementing a differentiable physics simulator is how to model contacts, as these tend to exhibit highly nonlinear and discontinuous behavior. These discontinuities arise as contacts exist in a binary state, as the objects can only ever be in contact or not. This can be seen in Fig. 4 where contact forces are discontinuous, step-like functions, where the gradient is not well-defined.

In the following, contact forces will be denoted as $\lambda = (\lambda_n, \lambda_t) \in \mathbb{R} \times \mathbb{R}^2$, where λ_n and λ_t are the normal and tangential forces relative to the contact surface between two objects. Similarly, relative velocities between two objects are represented as $\dot{q} = (\dot{q}_n, \dot{q}_t) \in \mathbb{R} \times \mathbb{R}^2$, where $\dot{q}_n$ and $\dot{q}_t$ are the normal and tangential velocities relative to the contact surface between two objects.

Contact models typically serve two purposes: to resolve interpenetrating objects, and to apply frictional forces. First, the contact model ensures that two solid objects in contact cannot penetrate through each other. This non-penetration requirement can be mathematically represented as

$$(\lambda_n, \dot{q}_n) \in \{(\lambda_n, \dot{q}_n) \in \mathbb{R} \times \mathbb{R} : \lambda_n \geq 0, \dot{q}_n \geq 0, \lambda_n \dot{q}_n = 0\}, \quad (10)$$

i.e. contact forces should always repel two objects instead of pulling them together, two objects cannot penetrate each other, and objects are either in contact (normal velocities are zero) or are not in contact (normal forces are zero). This last condition is known as a complementarity condition.

Second, frictional forces should be modeled when two objects are in contact with each other. Frictional forces in physics simulators are most commonly modeled using Coulomb's friction law. This states that the maximum

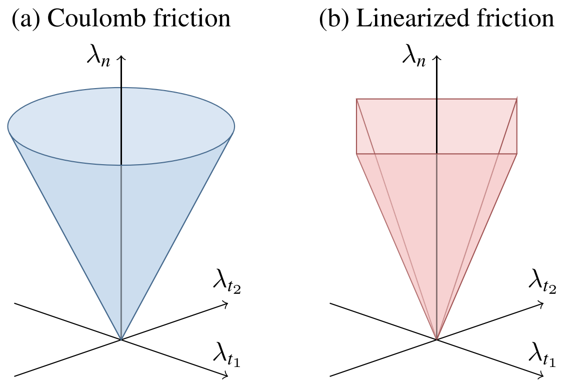


FIGURE 3. Comparison between (a) the second-order frictional cone defining Coulomb's law (see Eq. (11)), and (b) the square pyramid which linearizes the cone for LCP. Note that the linearized cone biases frictional forces towards the edges of the pyramid.

magnitude of the frictional force is proportional to the normal force, *i.e.*

$$\lambda \in \mathcal{K}_f = \{(\lambda_n, \lambda_t) \in \mathbb{R}_+ \times \mathbb{R}^2 : \|\lambda_t\|_2 \leq \mu\lambda_n\}, \tag{11}$$

where $\mu \geq 0$ is the coefficient of friction between two surfaces. Note that $\mathcal{K}_f$ is a (nonlinear) second-order cone. Additionally, friction must always act in the opposite direction to the tangential velocity, *i.e.*

$$\lambda_t = -\mu\lambda_n \frac{\dot{q}_t}{\|\dot{q}_t\|_2}, \quad \text{if} \quad \|\dot{q}_t\|_2 > 0. \tag{12}$$

We refer the reader to [86], [87], and [88] for further resources on contact models.

1) COMPLEMENTARITY PROBLEM

The most natural way to solve for contact forces is to find a suitable pair of forces and velocities that satisfy the non-penetration conditions and Coulomb's friction law. This involves solving the following feasibility problem

$$\text{find} \quad (\lambda, \dot{q}) \in \mathbb{R}^3 \times \mathbb{R}^3 \tag{13a}$$
$$\text{subj. to} \quad (10),\ (11),\ \text{and } (12). \tag{13b}$$

This is referred to as a nonlinear complementarity problem (NCP), where the nonlinearities arise from the frictional constraints. Some works solve this NCP, or variations of this NCP, directly [6], [36], [48]. Alternatively, some works [4], [26], [39], [42], [45] instead approximate the second-order friction cone (Fig. 3(a)) as a friction pyramid (Fig. 3(b)), which results in a linear complementarity problem (LCP). Although easier to solve, a drawback to this linearization is that frictional forces are biased towards the corners of the friction pyramid [86], [88].

To obtain gradients from the complementarity-based contact model, works which use optimization methods [6], [26], [48], such as interior-point methods, can derive the gradients by implicitly differentiating through the Karush-Kuhn-Tucker (KKT) conditions (see Section II-A3). The KKT conditions are necessary optimality criteria for constrained optimization problems. Another common way the complementarity problem can be solved is by using the projected Gauss-Seidel (PGS) method [4], [36], [39]. A simple method to obtain gradients through this method is to automatically differentiate through the PGS algorithm [4], [36], although, this can result in a large computational overhead. Conventional implicit differentiation techniques on the optimality conditions can avoid building this computational graph by directly obtaining explicit expressions for the gradient. However, Qiao et al. [39] notes that the PGS method does not guarantee that the complementarity constraints are satisfied when the algorithm is terminated, and therefore implicit differentiation techniques can produce inaccurate gradients. Instead, Qiao et al. [39] proposes a reverse version of the PGS method using the adjoint method (see Section II-A3) to obtain gradients, without requiring the computational overhead of automatic differentiation techniques. In Werling et al. [42], which solves the LCP using standard linear programming techniques, gradients are derived analytically without recasting it as an optimization problem.

Alternatively, in [7] and [13], the complementarity conditions are directly incorporated into the Projective Dynamics framework (see Section II-B2). The Projective Dynamics solver is augmented with an active-set method which iteratively searches for which constraints are active. Gradients are then obtained using a similar method to the Projective Dynamics framework without contacts (see the last paragraph of Section II-D2).

Related to the complementarity problem formulation, [37], [44] approximate the complementarity problem as a convex optimization problem (more specifically, as a quadratic program), this approach takes inspiration from the approach in MuJoCo [25]. Gradients are returned by implicitly differentiating through the optimality conditions of the convex optimization problem.

A related class of contact models to complementarity problems are impulse-based methods, which similarly account for contacts by changing state velocities. Impulse-based methods include [33], [41], [50], [53], [89], which model non-penetration requirements as an explicit impulse function which are easy to automatically differentiate over.

For these methods, it was identified in [5] that gradients at collision points can be inaccurate when the model is discretized in time. In particular, a naïve time discretization assumes that collisions only occur at discrete time intervals, and this inaccuracy worsens with larger time intervals. To tackle this issue, a continuous time-of-impact (TOI) method is proposed in [5] which computes the precise time at which a contact occurs, even if the contact occurs in between two discrete time intervals. In [87], the TOI method was shown to produce more accurate contact gradients for a variety of complementarity- and impulse-based contact models. Other variations of the TOI method are proposed by [53] and [90] aiming to produce more accurate gradients.

2) COMPLIANT MODELS

Complementarity-based contact models simulate "hard" contacts, *i.e.*, there is strictly zero penetration between objects. Another way to model the contact physics is to approximate the contacts as soft contacts, *i.e.*, some non-zero penetration between objects is allowed. Compliant models relax the step-like feature used for contact modeling (see Fig. 4) functions by approximating them with finite but sufficiently large gradients. This allows gradients to be obtained directly from these functional approximations of non-penetration and friction forces. The tradeoff is that contact models are no longer strictly enforced, *e.g.*, penetration is now modeled as a soft contact rather than a hard contact. The use of soft contacts can result in objects penetrating each other, however, this is normally aimed to be minimized.

The simplest approximation of non-penetration forces is to use a piece-wise linear function, *i.e.*

$$\lambda_n = k_n \max(-q_n, 0), \tag{14}$$

where $k_n \in \mathbb{R}_+$ represents the gradient of the linearization, and $q_n \in \mathbb{R}$ is the penetration distance between two objects (negative means they are penetrating). This can be interpreted as modeling non-penetration as a spring-like force, or as using a ReLU activation function. This linearization is used by [6], [7], and [56]. A more complicated contact model for fabrics is used in [46], which wraps a combination of bending and stretching forces inside a ReLU function. In [59] where objects are modeled as density fields, the non-penetration force is proportionate to the volume of interpenetration between the objects. A potential issue with such linearizations is that there is zero gradient when contacts are inactive, making it difficult for gradient based optimizers to make new contacts [51]. In [51] and [62], this is resolved by introducing a small gradient to the inactive portion of the compliant model, analogous to the leaky ReLU function. Apart from linear models of non-penetration forces, a quadratic approximation is used instead in [27], *i.e.*, $\lambda_n = k_n \max(-q_n, 0)^2$. Some works use a more complicated spring-damper model for normal forces, where $\dot{q}_n$ is also considered in the model [8], [36], [43].

For friction, Coulomb's law is commonly approximated as a piece-wise linear function, *i.e.*

$$\lambda_t = -\frac{\dot{q}_t}{\|\dot{q}_t\|_2} \min(\mu\lambda_n, k_t\|\dot{q}_t\|_2), \tag{15}$$

where $k_t \in \mathbb{R}_+$ determines the gradient of the linearization. This formulation is used by [6], [7], [8], [43], and [51]. More complicated models typically try to use a smoother approximation or model more complex dynamics not considered in Coulomb's friction law (*e.g.*, modeling different static and dynamic friction coefficients) [27], [36], [46].

3) POSITION-BASED MODELS

Whereas complementarity-based and compliant models determine the effect of contacts on forces and velocities, another method is to directly constrain the positions of objects to avoid penetrations. A commonly-used framework to model this kind of contact model is XPBD (see Section II-B2) which is used by both Warp [83] and Brax [35].

Other position-based models first perform an update step without considering contact physics and then perform a correction step to satisfy contact-based constraints. Turpin et al. [62] implements position-based dynamics based on [91], where the correction step is performed using a Gauss-Seidel method. Coulomb's friction can also be accounted for using this approach, as discussed in [91]. Alternatively, for cloth simulation, the correction step can be performed by using a quadratic program which minimizes the change in position of cloth particles while satisfying non-penetration constraints [29], [31]. Gradients are obtained from this optimization problem by using implicit differentiation on the KKT conditions. Both of these works use a QR decomposition trick to speed up the computation of gradients, taking advantage of the fact that the number of non-penetration constraints is typically much smaller than the number of simulation variables.

4) MLS-MPM

For the MLS-MPM dynamics model (see Section II-B2), which incorporates both rigid-body and soft-body objects, there are two types of contacts that are considered. First, self-collisions of soft-body objects are handled "automatically" by the MLS-MPM model, which accounts for momentum transfer between neighboring soft-body particles and grids. Second, collisions between soft-bodies and rigid-body obstacles are treated as boundary conditions, which are similar to the complementarity conditions given by Eqs. (10) and (11), and enforced by simply projecting the velocities into these sets after they have been updated. Gradients for these collision models are derived analytically in [14] and [61] to make the differentiable simulator as high-performance as possible.

D. INTEGRATOR

An integration scheme is required to use the computed forces to propagate a given dynamic model forwards in time. Throughout this section, the notation $q_k, \dot{q}_k, \ddot{q}_k \in \mathbb{R}^n$ represents the position, velocity, and acceleration state vectors at the k-th time step, respectively, and $\Delta t \in \mathbb{R}_+$ is the time step between each iteration.

Two main categories of integration schemes are explicit (Section II-D1) and implicit (Section II-D2) integration methods. Although explicit integration is easier to implement in practice, it is well known that it is less numerically stable compared to implicit integration schemes. For example, experiments in [7] show that implicit integration is stable at time steps of up to 10ms, whereas explicit integration is only stable at time steps of up to 0.5ms. A consequence of this is that explicit schemes have a larger memory overhead due to needing to simulate more time steps, and therefore requires tricks such as "checkpointing" to reduce this memory overhead [5], [39], [61]. Although explicit integration schemes

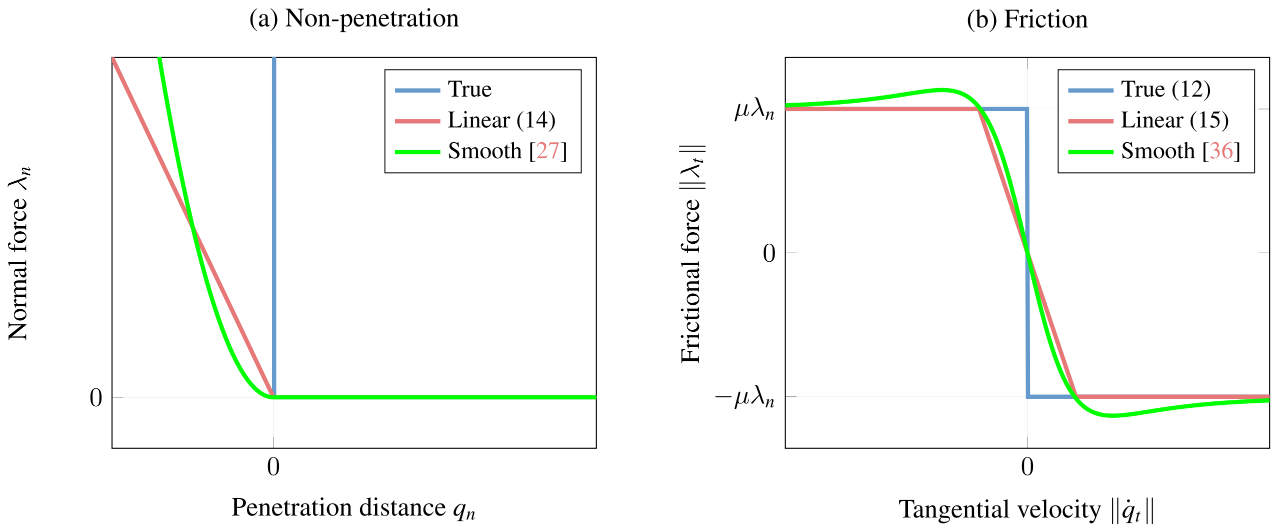


FIGURE 4. Comparison of different contact model implementations. The non-penetration requirements and Coulomb's friction law do not have well-defined gradients at $q_n = 0$ and $\|\dot{q}_t\| = 0$, respectively. Compliant models relax these models by approximating the discontinuities, which we can consider as impulse function-like gradients, using functions with finite but sufficiently large gradients.

are relatively straightforward to differentiate through, implicit integration schemes, which involve solving a nonlinear system of equations, require more complex techniques to extract gradients from. For both techniques, a standard way to improve the stability and accuracy is to use higher order integration methods, such as the Runge–Kutta family of integration schemes. We refer to, *e.g.*, [92] for a more in depth comparison of these integration methods.

1) EXPLICIT INTEGRATION

The simplest integrator, explicit (forwards) Euler integration updates an object's position and velocity by taking a small time step and applying the current velocity to the position and the current acceleration to the velocity, *i.e.*

$$\dot{q}_{k+1} = \dot{q}_k + \Delta t \cdot \ddot{q}_k \tag{16a}$$
$$q_{k+1} = q_k + \Delta t \cdot \dot{q}_k. \tag{16b}$$

This method is used by [9], [14], [26], [30], [32], [37], [39], [42], [47], [52], [58], [60], [61], and [64]. Explicit integration schemes are desirable as it is straightforward to propagate gradients through time just by using the chain rule. However, it is well-known that explicit integration is conditionally stable [93], [94], [95], *i.e.*, there exists a critical time step $t_{\text{crit}} > 0$ such that the simulation can be unstable if $\Delta t \geq t_{\text{crit}}$. This can lead to stability and accuracy issues, particularly with more complex simulations.

To obtain better stability properties, higher-order explicit schemes are used in [44], [50], [54], and [57], *e.g.* the explicit fourth-order Runge-Kutta method (RK4), which offers improved accuracy and stability compared to the basic Euler method by using multiple stages to update positions and velocities [96]. Other works [4], [8], [10], [27], [28], [35], [36], [41], [43], [45], [51], [62], [63] instead use semi-implicit Euler integration (symplectic integration), with the following steps

$$\dot{q}_{k+1} = \dot{q}_k + \Delta t \cdot \ddot{q}_k \tag{17a}$$
$$q_{k+1} = q_k + \Delta t \cdot \dot{q}_{k+1}, \tag{17b}$$

i.e. the velocity is first updated explicitly, then position is updated using this new velocity $\dot{q}_{k+1}$, rather than the current velocity $\dot{q}_k$, as in explicit integration. These semi-implicit steps are similarly easy to implement and backpropagate gradients through as explicit steps.

An issue shared by these methods is that, as discussed in [39], to backpropagate gradients through time, information at every time step needs to be stored in memory. This can be prohibitively expensive when the total simulation horizon is large, particularly as explicit integration schemes require small time steps to be stable. In [5], [39], and [61], checkpointing methods are proposed to reduce memory requirements of the simulator in exchange for slower runtime. During the forwards pass, information is "checkpointed" rather than storing the entire computational graph (*e.g.*, only store the simulation state for every n time steps). In the backwards pass, information required to compute the gradients that was lost can be recovered by re-running the simulation from the closest checkpoint.

2) IMPLICIT INTEGRATION

Implicit integration is designed to handle stiff differential equations and constraints more effectively than explicit methods, and therefore allows for much larger time steps to be used [93], [94], [95]. Implicit (backwards) Euler integration computes the state at the next time step by using the gradient at the next time step, *i.e.*

$$\dot{q}_{k+1} = \dot{q}_k + \Delta t \cdot \ddot{q}_{k+1} \tag{18a}$$
$$q_{k+1} = q_k + \Delta t \cdot \dot{q}_{k+1}. \tag{18b}$$

The implicit Euler method is used by [6], [7], [13], [29], [30], [31], [34], [46], and [56]. Other first order implicit methods have been used, such as [52] which uses a first-order backwards differentiation formula (BDF1). Like explicit methods, higher-order implicit methods can be used to improve stability and accuracy. For example, the second-order backwards differentiation formula (BDF2) is used in [11] and [38], and the implicit second-order Runge-Kutta (RK2) is used in [33] and [52]. In [53], only some of the state variables are updated using implicit integration, while the rest are updated using explicit integration, to reduce the computational cost of solving a large system of equations. In [48] and [59], a variational integration scheme is used, where instead of discretizing the system dynamics as in Eq. (18), the integration scheme is derived by discretizing Hamilton's Principle. This results in an integration scheme which conserves momentum and energy, and is therefore stable even for relatively large simulation time steps.

Unlike explicit Euler, implicit Euler steps represent a set of (possibly) non-linear equations which must be solved to find the state at the next time step. A standard way to solve these equations is to use an iterative method, such as Newton's root-finding method [6], [11], [48]. Alternatively, some works take a first-order Taylor-series expansion of the equations [29], [31], [46]. In this case, solving for the state at the next time step involves a simple factor-solve of the resulting linear system of equations. For both of these methods, the gradient can be computed by implicitly differentiating through the equations (see Section II-A3). The adjoint method is commonly used here to make the gradient computation more efficient (see Section II-A3).

Alternatively, in [7] and [13], Projective Dynamics (see Section II-B2) is used to express the nonlinear set of equations Eq. (18) as an energy minimization problem. By making certain assumptions on the system dynamics, this optimization problem can be solved by using an alternating optimization algorithm. These steps can be cheaper than a naïve Newton's method by taking advantage of the fact that the Cholesky factorization of one of the matrices arising from the alternating optimization scheme can be precomputed. The gradient backpropagation step, which is still derived by implicitly differentiating the optimality conditions, can also take advantage of the same precomputed Cholesky factorization. Numerical experiments in [7] demonstrate the computational superiority of the Projective Dynamics approach compared to naïve implicit differentiation implementations.

III. DIFFERENTIABLE SIMULATORS

This section aims to be a more practical guide to differentiable simulation engines. Throughout the literature on differentiable simulators, eight differentiable simulators were identified [5], [30], [35], [36], [42], [48], [58], [83] that were adopted by other works included within the scope of this review. Table 2 summarizes the key features of these eight engines and three additional differentiable physics engines. While the latter three have been recently proposed and not yet widely adopted, they show promise for practical applications. Most of the simulator codebases focus on rigid body simulation [35], [36], [42], [48], [83], however, some do support soft body simulation [5], [83] and fluid simulation [30], [57], [65].

A. WARP

NVIDIA Warp [83] is a Python framework designed for high-performance simulation and graphics code, featuring just-in-time (JIT) compilation of Python functions for efficient CPU and GPU execution.

Warp allows for both soft and rigid body simulation, as well as supporting two types of integrators, semi-implicit and XPBD [72]. Importing can be done by adding joints manually or through importing either a URDF, MJCF or USD. It supports non-differentiable rendering and basic differentiable raycasting to visualize the model and is based on a maximal coordinate system. Warp supports triangular-based meshes which can be created through the API by providing points, indices and velocities. Most of the underlying functionality in Warp has featured in other Nvidia differentiable simulation engines, such as gradSim [8], before being distilled as Warp.

B. TINY DIFFERENTIABLE SIMULATOR

Tiny Differentiable Simulator (TDS) [36] is a C++ and CUDA physics library, known for its simplicity and independence from external dependencies. TDS focuses on rigid-body dynamics and contacts, therefore, TDS does not support deformable objects. A limitation of the simulator is its support for only primitive collision shapes, like spheres, planes and capsules. One key advantage of TDS is its ability to run thousands of parallel simulations on a single GPU. TDS does not have detailed documentation describing the API, however, it does have a number of comprehensive examples.

C. NIMBLE

Nimble [42], stemming from a fork of the popular DART physics engine [97], has evolved into a fast and fully differentiable physics engine with analytical gradients and PyTorch bindings. Nimble supports non-differentiable browser-based rendering of the simulation scene. However, Nimble was designed for single-threaded execution and does not support parallel environments.

D. DIFFTAICHI

DiffTaichi [5], implemented using the Taichi [98] differentiable programming framework, supports both rigid-body and soft-body simulations. Several standalone environments exist in DiffTaichi operating as examples of capability, however, unlike other simulators within this section DiffTaichi is not intended a general purpose physics engine, rather aiming to provide a method to calculate gradients of dynamical systems effectively and efficiently through source code transformations [99]. As such, DiffTaichi does not support

TABLE 2. A table featuring eleven open-source differentiable simulators. Eight of these simulators have been utilized in existing literature reviewed herein, while the remaining three (below the bolded line) have not yet been adopted due to their recent introduction. This is a subset of the engines found in Table 1, with a focus on user features researches may consider when choosing a differentiable simulator. All engines have Python bindings and can be run on GPU resources. There are three types of objects supported: Rigid, Deformable (Def) which includes cloth and Fluids.

	Object Types				Integrations		File Types Supported				Collision Type				
Engine	Rigid	Def	Fluids	Language	Jax	PyTorch	URDF	MJCF	USD	Parallel Sim	Primitive	Mesh	Particle	SDF	Diff Render
TDS [27]	✓			C++			✓			✓	✓				
Warp [83]	✓	✓		Python	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DiffTaichi [5]	✓	✓		C++	✓	✓							✓		
Brax [35]	✓			Python	✓		✓	✓		✓	✓	✓			
Nimble [42]	✓			C++		✓	✓				✓	✓			
Dojo [48]	✓			Julia	✓	✓	✓				✓				
GradSim [8]	✓	✓		C++		✓	✓		✓		✓	✓			✓
PhiFlow [30]			✓	Python	✓	✓									
daX [58]	✓	✓		Python	✓	✓	✓	✓	✓	✓	✓	✓			
FluidLab [65]	✓	✓	✓	Python		✓					✓	✓	✓	✓	
JAX-Fluids [57]			✓	Python	✓										

common file types, such as URDF, MJCF or USD, and kinematic chains needs to be entered pragmatically via the API.

E. BRAX

Brax [35], which is based on JAX [68], allows for scalability across parallel simulations. Brax supports several backend physics pipelines and supports both MJCF and URDF file loading as well as programmatically creating object models. Brax supports a non-differentiable browser rendering system.

F. GradSim

GradSim [8] (∇Sim) aims to unify the growing popularity of differentiable rendering [100] with differential physics. GradSim supports simulation of both rigid and deformable objects. However, GradSim does not support parallel execution or include documentation about the API, this could increase the learning curve for new users.

G. DOJO

Dojo [48] is a differentiable physics engine tailored for robotics. Dojo uses a variational integrator for stability and to help with handling large timesteps. Dojo does not support mesh-based collisions, only allowing collisions between primitive shapes. Dojo is no longer actively maintained, however, there is evidence of ongoing adoption of this engine throughout the literature.

H. PhiFlow

PhiFlow (ϕFlow) [30] aims to create a differenitable simulation framework for solving partial differential equations (PDE), mostly focusing on fluid simulation applications. It supports both Pytorch [67] and JAX [68], as well as a non-differentiable HTML-based rendering. Objects can be loaded from *su2* [101] CFD files, which is an open source CFD simulation software.

I. ADDITIONAL DIFFERENTIABLE SIMULATORS

Some differentiable simulators have recently been proposed and although they have not been adopted by other research, they are worth noting. DaXBench [58] is one such simulator. DaXBench supports various deformable objects like fluids, ropes, and cloth. daX uses JAX [68] and integrates seamlessly with OpenAI Gym [102]. DaXBench aims to serve as a tool for standardized testing and development of new approaches in deformable object manipulation using differentiable simulation.

FluidLab [65] is another such differentiable simulator and is tailored towards fluid manipulation task, but also supports rigid and deformable object manipulation. The differentiable simulator is called FluidEngine and is built using the Taichi [5] differentiable programming paradigm. The focus of FluidLab is 10 distinct manipulation tasks, where the agent has to interact with certain types of fluids to achieve a goal.

JAX-Fluids [57] is the final noteworthy inclusion. JAX-Fluids is written in JAX [68] and designed for CFD. It incorporates state-of-the-art numerical methods and is intended to allow integration of CFD within machine learning workflows. JAX-Fluids aims to handle complex fluid dynamics scenarios, including three-dimensional turbulence, compressibility effects, and two-phase flows. The configurations for the simulator rely on the JSON format but loading objects within the simulator is currently not supported. The simulator does not currently support parallel simulations, however, the authors have stated their intention to support this capability in the near future.

IV. APPLICATIONS

In the literature, differentiable simulators have been applied as a solution for many problems. These applications of differentiable simulators all fundamentally involve solving an optimization problem, which are typically solved using gradient-based optimization algorithms. For an overview of these optimization algorithms, we refer the reader to [103].

We review five primary types of application domains:

1) System Identification – leverages differentiable simulators to model complex systems, allowing for efficient parameter tuning and system characterization.
2) Trajectory Optimization – employs differentiability to refine and optimize the path of objects or entities within a simulated environment, enhancing control capabilities.
3) Morphological Optimization – utilizes differentiable simulators to evolve and optimize the physical structures or morphologies of entities. Typically, morphology optimization is undertaken as co-optimization of both morphology and control.
4) Policy Optimization – uses differentiable simulators to train and optimize policies, particularly in a reinforcement learning framework, facilitating the development of robust and adaptive decision-making strategies.
5) Neural Network Augmented Simulation - integrates neural networks within the simulator to closer align simulations to the real-world.

The application areas of differentiable simulators listed above are categorized based on the typical distinctions made in literature. All papers that investigate use cases of differentiable simulators can be categorized into one of these five application areas. The number and overlap of works in each application domain are summarized in Fig. 5.

A. SYSTEM IDENTIFICATION

System identification aims to construct an accurate mathematical model of a physical system by estimating unknown parameters based on observations of the system. To estimate the model parameters $\theta \in \mathbb{R}^p$, most works minimize the mean squared error between the observed state trajectory $q_1, \dots, q_N \in \mathbb{R}^n$, and the state trajectory estimated by the physics simulator $\hat{q}_1(\theta), \dots, \hat{q}_N(\theta) \in \mathbb{R}^n$ given the model parameters θ, *i.e.*,

$$\min_{\theta} \sum_{i=1}^{N} \|\hat{q}_i(\theta) - q_i\|_2^2 \tag{19a}$$

$$\text{subj. to} \quad \theta_{min} \leq \theta \leq \theta_{max}, \tag{19b}$$

where $\theta_{min}, \theta_{max} \in \mathbb{R}^p$ are the minimum and maximum allowable values of the parameters.

System identification has been used to estimate the inertial properties (*e.g.*, mass and moment of inertia) of colliding objects or objects given an initial known impulse [8], [14], [26], [31], [33]. Similarly, inertial and kinematic properties (*e.g.*, length) of a pendulum-like system have been estimated in [15], [28], and [105]. Other works estimate the stiffness and elasticity properties (*e.g.*, Young's modulus, Poisson's ratio) of cloth [13], [29], [46], [106], bouncing objects [7], [44] (see Fig. 6), rope [60], or actuated soft robots [61], [107], [108]. Other common applications include estimating the coefficients of friction of robots [47], [89] or other objects sliding on a surface [32], [37], [59], and to identify a recorded trajectory of a robot [50], [109] or human [45]. In [10], it was shown how differentiable simulators could be used to estimate properties of a knife and a soft object that it was cutting, as well as the cutting force being applied. In [110], [111], URDF models of household objects which are initially obtained from a scanned point cloud are iteratively improved using differentiable simulation, by using trajectories observed from a robot agent interacting with the object. Some works perform system identification using differentiable rendering techniques [8], [50], [105], [106], [108], [110], where the loss function Eq. (19a) is instead defined as the pixelwise mean square error between a recorded video of the true system, and a rendered video from the differentiable simulator. Many works apply system identification to estimate parameters of real-world systems [10], [13], [15], [28], [32], [33], [34], [37], [47], [50], [59], [60], [74], [89], [105], [106], [108], [110], [111], [112].

Advantages and Challenges: Compared to black-box or gradient-free methods for system identification, the use of differentiable simulations has been shown to converge to lower cost solutions in less computational time [8], [13], [29], [44], [47], [106]. Gong et al. [46] showed that differentiable simulators enabled accurate system identification of cloth parameters by only using a small amount of recorded time steps, which only contains subtle movements of the cloth. This is as opposed to other approaches which were unable to accurately estimate the cloth parameters using the same limited set of data. In [7] and [37], it was noted that by nature of the formulation of the optimization problem Eq. (19), it is possible to obtain parameters which are quite different from the ground truth values, while still yielding similar predicted trajectories. This problem is called "parameter observability" in [37]. The authors show how this is a larger issue when there are more parameters that are estimated simultaneously, and how data of more complex trajectories (*e.g.*, collisions with objects with known parameters) can help to circumvent this issue by exposing certain desired parameters of the system.

System Identification and Control: One of the main uses of system identification is for the estimated parameters to be used for control tasks [15], [32], [34], [41], [47], [74], [89], [105], [110], [111], [112], *i.e.*, for use with trajectory optimization (see Section IV-B), policy optimization (see Section IV-D), or with more classical control methods. For example, in [15] it was shown how system identification allowed a robot to learn how to perform the complex "ball-in-cup" task, whereas black-box methods failed to perform the same task. In this control context, system identification has also been performed in an online fashion, where system identification is performed on continuously measured trajectory data while an agent is simultaneously being controlled to perform a task. This has been shown to be advantageous for dynamically changing environments [47], [74], to continuously improve the accuracy of the estimated parameters [89], or to avoid the need for collecting offline data beforehand [34]. In [74], a trajectory

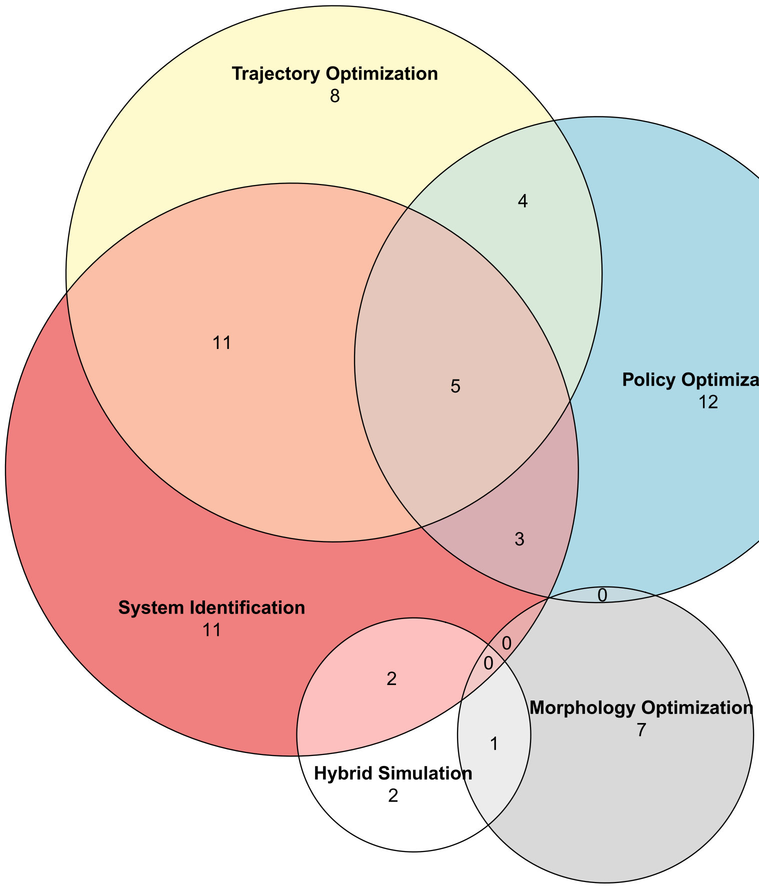


FIGURE 5. An Area-Proportional Venn diagram [104] visualizing the different application areas of differentiable simulators explored in this review with the numbers indicating the number of referenced works. The number of cited works for a given application area in the Venn diagram is cumulative.

buffer system was used to account for noisy and outdated observations, which would otherwise lead to inaccurate or unstable parameters estimations.

B. TRAJECTORY OPTIMIZATION

Trajectory optimization aims to determine optimal sequences of states and control inputs for dynamic systems. The goal is to minimize a predefined cost function, typically composed of a per-step loss function $\mathcal{L}(u_i, \hat{q}_i)$ and a terminal cost $\mathcal{L}_{\text{terminal}}(u_N, x_N)$. Here, $u \in \mathbb{R}^m$ represents control parameters, and $\hat{q}_1(u), \ldots, \hat{q}_N(u) \in \mathbb{R}^n$ denote the state trajectory estimated by the physics simulator given control parameters u *i.e.*,

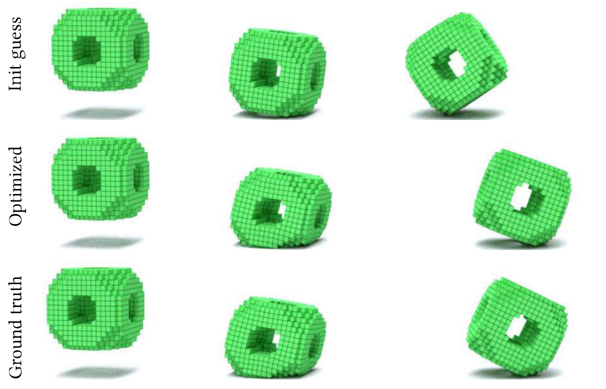
$$\min_{u} \sum_{i=1}^{N-1} \mathcal{L}(u_i, \hat{q}_i) + \mathcal{L}_{\text{terminal}}(u_N, q_N) \tag{20a}$$

$$\text{subj. to} \quad u_{min} \leq u \leq u_{max}, \tag{20b}$$

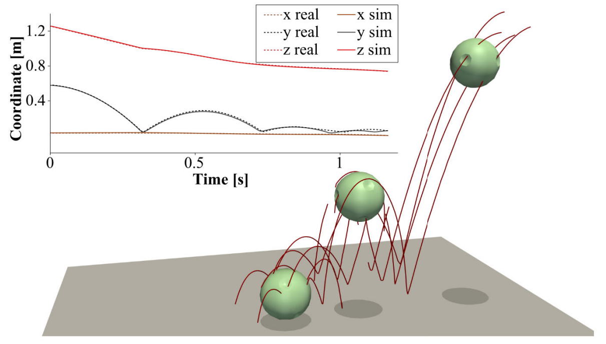
where $u_{min}, u_{max} \in \mathbb{R}^m$ represent the minimum and maximum allowable values of the control parameters. This definition aligns closely with the system identification problem, however, focusing on optimizing the control signal under the assumption of known model parameters. Trajectory optimization has commonly been coupled with other tasks such as system identification [15], [34], [41], [47], [74], [89] and morphology optimization [11], [14]. For example, [34] combines system identification and trajectory optimization, by simultaneously updating both the trajectory and the model (against real-data) iteratively until a controller can produce large forward velocities (see Fig. 7, right).

Common Tasks: A common trajectory optimization task involves moving a simulated body in a specified direction [7], [14], [53], [89], with the aim of either maximizing speed or maintaining a fixed speed. This task is applicable to various body types, such as soft bodies [14], rigid-bodies [7] and tensegerity robots [53], [89]. Besides moving in a fixed direction, studies have explored other simple motions like jumping [42] or lifting a leg on a quadruped robot [75]. Wang et al. [63] presented benchmark tasks for the control of soft body objects, including moving forward, turning, tracking a velocity and following a set of waypoints.

Exploring the dynamics of throwing objects has been investigated in the context of robotic manipulators [6], [44]. A related task involves manipulating fabrics, where studies focus on placing a cloth into a bucket by applying forces to its corners [31], [46]. Additionally, Li et al. [13]



(a) System identification of a simulated bouncing ball [7]. Shown are the initial frames (left), the moment the ball collides with the ground (middle), and final frames (right) of trajectories arising from randomized (top), optimized (middle), and ground truth (bottom) material parameters.



(b) System identification of a real-world bouncing ball [6]. The red lines show the motion-captured trajectories of the ball, and the green ball shows the estimated trajectory of the ball with optimized material parameters. The graph compares the real-world and estimated trajectories of the ball.

FIGURE 6. Examples of applications of differentiable simulation to perform system identification. Both examples estimate the material parameters (*e.g.*, stiffness and dampening ratio) of an elastic bouncing ball so that the trajectory of the ball matches the ground truth as closely as possible.

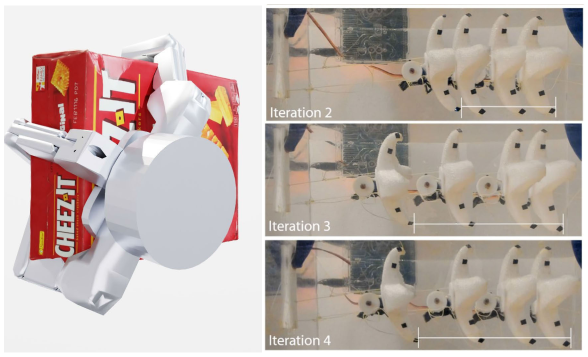


FIGURE 7. (Left) Turpin et al. [62] optimize a grasp pose for various metrics. (Right) Du et al. [34] use trajectory optimization to control the starfish to move forwards. The results are shown over several iterations. (© 2024 IEEE).

investigated cloth-based tasks, specifically a dressing task using a manipulator to put socks and hats onto a simplified model of a human body.

Various manipulation-based tasks have also been explored, including reaching using muscle driven arms [52], trajectory optimization for specified paths [6], [50], object pushing [32] and batting a ball towards a target [42]. Turpin et al. [51] focuses on object grasping, optimizing hand poses through trajectory optimization. The approach in [51] only requires the final pose after optimization, and this is extended in [62] to enhance both speed and stability (see Fig. 7, left). Le Cleac'h et al. [59] explores robot trajectory optimization with contact interactions, specifically in a push-and-slide task where a simulated robot arm moves an object (Stanford bunny) to a goal while returning the robot's end effector to another goal, utilizing Dojo [48].

Lin et al. [113] focus on tool-assisted manipulation tasks with soft bodies, like lifting and spreading dough on a cutting board, followed by flattening it with a rolling pin. In a similar context, Li et al. [12] use a robotic hand to manipulate dough towards specific goals. Generalizing the scope, Huang et al. [9] present benchmarks for soft-body manipulation tasks involving agents reshaping plasticine material using manipulators.

Comparisons Against Other Methods: Huang et al. [9] demonstrate that gradient-based methods, by optimizing open-loop control sequences using differentiable physics engines, efficiently solve benchmark tasks, outperforming reinforcement learning (RL)-based approaches. Similar findings are reported in other works, including [11], [14], [29], [31], and [46], highlighting the effectiveness of trajectory optimization compared to RL algorithms like proximal policy optimization (PPO) [114]. However, some tasks, such as those in Li et al. [12] and considered by Lin et al. [113], suggest that RL might outperform trajectory optimization, especially in scenarios with longer time horizons. Both studies recognize the value of differentiable simulation, whether in refining trajectories [12] or aiding in data collection [110], [113]. Comparisons to derivative-free optimization techniques, commonly CMA-ES [115], consistently show that gradient-based optimization achieves solutions faster and often produces superior results [6], [11], [31], [42], [52].

While trajectory optimization requires more computation time compared to RL, which can generate real-time reactive policies, a strategy to enhance its speed is to combine it with imitation learning over an optimized trajectory. Chen et al. [116] integrate a differentiable physics simulator into the policy learning computational graph, aiming to minimize the divergence between expert and agent trajectories. Consequently, this RL-based method aims emulate the characteristics of trajectory optimization while still offering real-time solution.

C. MORPHOLOGY OPTIMIZATION

Morphology optimization aims to find a set of material and geometric properties of a system that optimizes predefined objectives, subject to physical and geometric constraints. There are parallels between morphology optimization, system identification and trajectory optimization, since morphology optimization can either identify the optimal morphological parameters of the system with respect to a fixed trajectory [27] or optimize them together with control parameters to generate a co-optimized trajectory [11], [14], [117]. The morphological parameters $\omega \in \mathbb{R}^p$, where p is the number of morphological parameters, encompass physical characteristics such as shape, size, material properties, joint configurations, and other structural features that define the morphology of the agent under consideration.

Co-Optimizing Morphology and Control: A common application of morphology optimization is in co-design tasks, formulated as simultaneously optimizing the structural design of a robotic system and minimizing the cost of a control policy which drives the system towards a desired goal state. It is typical to minimize a per-step loss function $\mathcal{L}(u_i, \hat{q}_i, \omega)$ and a terminal cost $\mathcal{L}_{\text{terminal}}(u_N, q_N, \omega)$, over both the control parameters u and the morphological parameters ω, and the state trajectory estimated by the physics simulator $\hat{q}_1(u, \omega), \ldots, \hat{q}_N(u, \omega) \in \mathbb{R}^n$ given both the control and morphological parameters, *i.e.*,

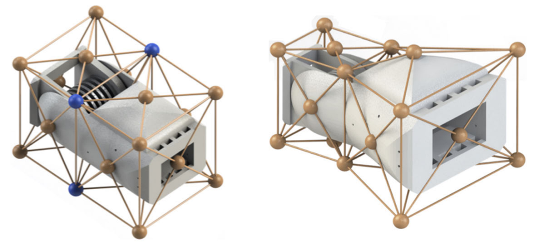
$$\min_{u,\omega} \sum_{i=1}^{N-1} \mathcal{L}(u_i, \hat{q}_i, \omega) + \mathcal{L}_{\text{terminal}}(u_N, q_N, \omega) \tag{21a}$$

$$\text{subj. to} \quad \omega_{min} \leq \omega \leq \omega_{max}, \quad u_{min} \leq u \leq u_{max}, \tag{21b}$$

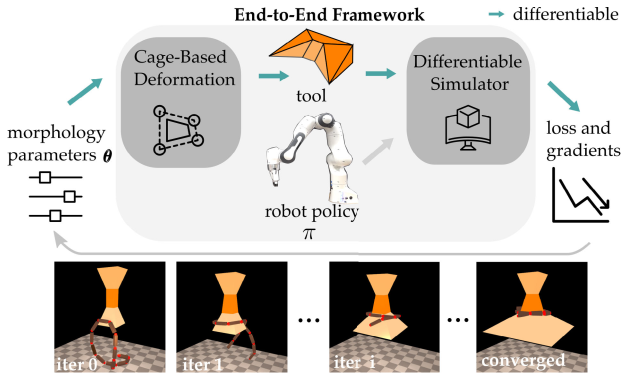
where $\omega_{min}, \omega_{max} \in \mathbb{R}^p$ and $u_{min}, u_{max} \in \mathbb{R}^m$ are the minimum and maximum allowable values of the morphological and control parameters respectively. Here, the loss function closely follows the definition of that for a typical trajectory optimization problem, however, the co-design task is defined over both the control parameters u and the morphological parameters, ω.

Differentiable simulation offers a means to overcome the *curse of dimensionality* typically associated with such problems by allowing gradient information to be used when solving the problem. Because of the ability to make difficult problem spaces tractable, differential co-design [118] is a popular methodology for optimizing morphology and control of soft robots.

Exemplar tasks include finding the optimal distribution of material stiffness (Young's modulus) across a deformable robot while minimizing the actuation energy to achieve a target pose [14], and optimizing the design of robot end-effectors and customized tools to enhance performance in contract-rich object manipulation tasks [11], [117] (see Fig. 8). Other works have explored using a combination of gradient-based methods and graph search in the morphological space to optimize the design of autonomous underwater and aerial vehicles [54], [55] for performing highly-dynamic motions in simulated surveying tasks. Heiden et al. [27] investigate the problem of optimizing the Denavit-Hartenberg parameters of a 4-DOF robot arm, given a pair of joint-space and task-space trajectories generated from an unknown kinematic mapping. Additionally, Mezghanni et al. [119] use the simulation gradients with respect to shape morphologies to train a neural network which generates more physically-stable shapes.



(a) Visualization of the morphology optimization process of a robot joint and body segment from [11]. (Left) Lower-dimensional morphology parametrization is constructed prior to optimization by posing deformation constraints on each component and joining their handle points (highlighted in blue) to ensure a feasible optimized morphology (Right).

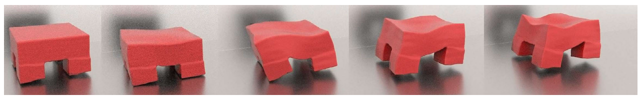


(b) Overview of the end-to-end morphology optimization pipeline from [117] leveraging differentiable simulation, where the tool morphology uses a similar caged-based parametrization as [11]. At each iteration, the robot executes the task using the current tool parameters in simulation, after which gradients of the simulation are used to update the tool morphology parameters for the next iteration. (© 2024 IEEE)

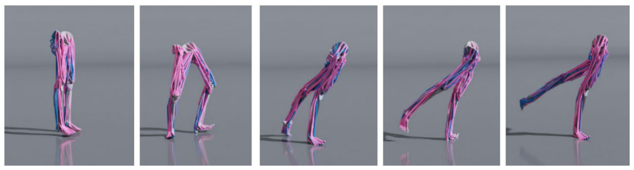
FIGURE 8. Examples of differentiable simulation applied to morphology optimization. Both examples utilize caged-based morphology parametrization.

Advantages and Challenges: Numerous works have found the use of differentiable simulators in co-design tasks to yield faster convergence and better constraint satisfaction [11], [14] compared to evolutionary algorithms and RL methods. Li et al. [117] show the robustness of the learned tool morphology on a sequence of manipulation tasks used in training with different initial conditions, however highlight the challenge of generalizing to other unseen tasks. Another drawback comes from the existence of local minima even in simple morphological design-spaces [63].

Improving the Optimization Process: An existing method of enhancing the efficiency of gradient-based morphology optimization is to alter the parametrization model. Cage-based parametrization [120] is a technique in computer graphics which uses vertices of a coarse, closed cage to control the enclosed space's deformation (see Fig. 8). This method has been shown to result in much lower-dimensional optimization spaces [11], [117] compared to mesh-based counterparts, enabling gradient-based methods to more efficiently generate smoother morphology with comparable task performance than solutions obtained from mesh-based parametrizations.



(a) A soft-body 3D quadrupedal agent with 16 actuators, 4 per leg, learns a running policy using APG [14].



(b) Visualization of a policy learned using a differentiable simulator to control 152 muscle-tendon units to accomplish a running gait [43].

FIGURE 9. Examples of policies learnt using gradients available from differentiable physics simulators. In both cases the policies are parameterized indirectly with either a simple perceptron or a neural network.

Through exploring the relationships between morphology representation, co-design tasks, and optimization algorithms, Wang et al. [63] find that having a prior distribution in the design-space can further improve the effectiveness of the optimization.

D. POLICY OPTIMIZATION

In contrast to trajectory optimization (Section IV-B), a control policy abstractly represents actions conditioned on an environment's state [121]. However, policy optimization has received less attention in the context of differentiable simulation due to inherent local minima and discontinuities in the optimization landscape [43]. Direct parameterizations of control policies include optimizing the angular frequency parameter within a parametric curve [49] and the parameters of a sinusoidal wave [10]. In the first example where the policy was conditioned on the angular frequency, the policy governed the motion of a soft carangiform swimmer while the parameterisation of the sinusoidal wave controlled the cutting action of a knife. While successful for specific problems, representing control policies as parametric curves or sinusoidal waves is only feasible for select problems. For most tasks, neural networks offer greater flexibility but introduce complexity in the loss landscape.

Analytic Policy Gradients: Some works optimize neural network parameters directly to encode control policies, termed Analytic Policy Gradients (APG) or policy optimization using Back Propagation Through Time (BPTT) [64]. Examples of this are typically constrained to environments with limited to no variation in start state, open-loop control, simplified environmental contacts, and short horizons [4], [6], [7], [13], [14], [31], [46], [61], [122], [123] (see Fig. 9 for an example). In closed-loop control tasks with long horizons or many contact-rich scenarios [113], [124], simple policy optimization methods like APG may yield suboptimal solutions. Therefore, various approaches are explored to find high-performing policies in such settings.

Enhanced Policy Learning: Qiao et al. [39] proposed two techniques to enhance policy learning with differentiable physics engines: sample enhancement and policy enhancement. Sample enhancement leverages first-order approximation of gradients from a differentiable simulator to generate additional accurate samples, aiding the critic in learning a value function. However, with the advent of highly parallelized differentiable simulators [83], this approach may be less valuable than parallel data collection. In policy enhancement, the policy network is updated using physically-aware gradients from the differentiable physics simulator, which are generally more accurate. These methods were demonstrated on a pendulum control task and the classic ant control problem, showcasing significant improvements over state-of-the-art RL algorithms.

Policy Optimization via Differentiable Simulation (PODS) [38] extends Deterministic Policy Gradient (DPG) RL algorithms. It incorporates derivatives from a differentiable simulator by computing the analytic gradient of a policy's value function with respect to its actions. This allows for the refinement of trajectory actions, leading to improved estimates of the value function. The policy network can thus be updated using the difference in the initial and the improved trajectory. Both first- and second-order policy improvement are proposed, involving a line search for stability and, for the latter, computation of a Hessian, which can be computationally intensive. Reported results demonstrate faster convergence to high rewards compared to other state-of-the-art (SOTA) methods on pendulum and cable-driven manipulation control tasks.

Short Horizon Actor Critic (SHAC) [43] is an RL algorithm leveraging gradients from a differentiable simulator over short time sequences, demonstrating performance improvements compared to SOTA RL algorithms (see Fig. 9(b)). The authors showcase SHAC's speed and sample efficiency on tasks including cartpole, ant, humanoid, and humanoid muscle-tendon unit (MTU), outperforming SOTA RL algorithms, first-order PODS, and BPTT in both wall time and sample efficiency. The method's success is credited to the GPU-based differentiable physics engine, enabling efficient simulation and accurate gradient updates to the actor within a truncated learning window, resulting in a smooth approximation in the loss landscape. However, when applied to the deformable manipulation tasks in DaxBench, SHAC was empirically worse than proximal policy optimization (PPO) and was often outperformed by APG.

Huang et al. [125] also proposes to do RL utilizing gradients from a differentiable simulator. However, uniquely they formalize the RL policy as a generative model over the distribution of optimal trajectories and train it using a variational lower bound optimized with gradients from both the classical policy gradient theorem and the differentiable simulator. The method's performance is only qualitatively evaluated on toy problems.

Policy Learning using Imitation Learning: In the domain of Imitation Learning, several works leverage differentiable physics simulation for trajectory optimization, with Behavior Cloning used to learn a policy parameterized as a neural network from expert trajectories [110], [112], [126]. In Chen et al. [116], gradients are computed using an expert trajectory and a neural network policy where a Chamfer-α loss is proposed that calculates the distance between an expert trajectory and the rolled-out policy trajectory.

Improving Convergence: Several methods have been proposed to enhance convergence in challenging optimization landscapes, yet their application to complex closed-loop control environments remains unexplored. Randomized smoothing [75] addresses non-smoothness and non-convexity issues by using a noise distribution to evaluate gradients near the current optimization stage. Similarly, an α-order gradient estimator [127] interpolates between zeroth and first-order gradient estimates to tackle nonlinearities and discontinuities in differentiable physics, preventing convergence to suboptimal policies.

BO-Leap [124] enhances a global search method with local search and gradient-based optimization, combining Bayesian Optimization, CMA-ES-like local search, and gradient-based optimization using differentiable simulator parameters. While effective in overcoming some discontinuities, it may struggle with non-smooth landscapes. Successful applications on tasks like cartpole and soft-body manipulation have been demonstrated, though the learned policies in these cases are open-loop with direct parameterizations.

Summary: In summary, most policy optimization methods attempt to learn neural network-parameterized control policies, with approaches like APG being common but prone to failure in increasingly complex tasks due to discontinuities and non-smoothness. While PODS and SHAC address some of these challenges, their robustness across diverse tasks remains unclear, leaving the integration of gradients from differentiable simulation for policy optimization an open question. Imitation learning approaches are often akin to trajectory optimization with behavior cloning. Promising yet under-explored directions involve incorporating randomized smoothing, adaptive interpolation of first and zeroth-order gradients, and augmentating gradient-based methods with global search algorithms to learn robust policies.

E. NEURAL NETWORK AUGMENTED SIMULATION

Neural network augmented simulation involves the integration of a neural network and a differentiable physics engine for improved alignment between simulation and the real-world, while keeping the overall augmented simulation pipeline differentiable. Lv et al. [111] trained a neural network to learn the residual between the real future state and the simulated next state from the physics engine (Fig. 10, left). Other works have explored embedding a differentiable physics engine within an autoencoder architecture, enabling end-to-end training of the overall network. This has been applied to system identification [26], [27], where object states decoded from input RGB frames are propagated forwards by a small timestep through the physics engine, before being encoded to generate output frames. These works have demonstrated the advantage of neural network augmented simulation resulting in better alignment between real and simulated environments.

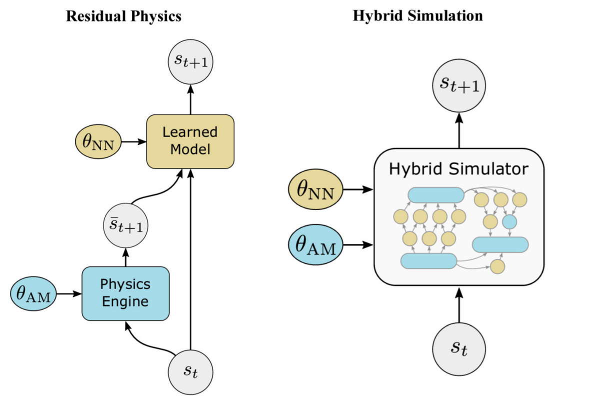


FIGURE 10. Comparison of neural network augmented simulation methods from [36]. (Left) The neural network learns the residual between the real states at the next timestep and simulated states from the physics engine. (Right) Hybrid simulation enables parts of the physics simulation to be replaced or augmented by neural networks, which can learn dynamics which the analytical physics models do not account for. (© 2024 IEEE).

Others used a neural network to learn a model of key physical parameters such as frictional contact dynamics [36] and physical properties of objects [26] (Fig. 10, right). These works have shown the augmented simulation approach to yield faster convergence on estimated system parameters that are closer to ground-truth, enabling better generalization and more accurate long-term rollout predictions than deep-learning approaches, while accounting for dynamics that the physics engine does not model [36].

V. DISCUSSION

The increasing interest in differentiable simulators primarily arises from the current demands of machine learning paradigms, which heavily rely on gradients for optimization. The proliferation of efficient gradient computation techniques and an improved understanding of gradient-based optimization algorithms have further fueled this trend. As detailed in Section III, there exists a strong array of differentiable simulators, serving as a valuable resource for newcomers interested in exploring the capabilities of these physics simulators or for integration into existing workflows.

The applications of differentiable simulators, as explored in Section IV, encompass a diverse range of areas including the optimization of simulation parameters such as actions, dynamics, and morphologies, as well as integration into broader differentiable workflows. However, the journey toward fully realizing the potential of these simulators is marked by notable challenges. Suh et al. [127] illuminate several critical limitations of current differentiable simulators. In complex physical systems, particularly those involving long-horizon planning and control, the performance of differentiable simulators is hindered by landscape characteristics such as non-linearity, non-smoothness, and discontinuities. This complexity underscores the necessity of nuanced approaches in gradient estimation and optimization.

Moreover, using deterministic gradients from these simulators can lead to sub-optimal behavior due to the inherent landscape characteristics. The phenomenon of 'empirical bias', where approximations of discontinuous dynamics lead to inaccuracies in gradient estimates, presents another significant challenge. Additionally, high variance in gradient estimates can be problematic, especially in scenarios with persistent stiffness or chaotic dynamics.

These complexities not only highlight the necessity for thoughtful design and algorithmic considerations in developing differentiable simulators but also underscore the importance of ongoing research and refinement in this evolving field.

A. FUTURE DIRECTIONS

Looking ahead, the path for differentiable simulators includes several promising areas of research and development. Some of these areas are application driven whilst others call for further research into the accuracy of simulation whilst enhancing the availability of meaningful gradients.

We see the following research directions as being areas of growth and continued development for differentiable simulators:

1) BETTER GRADIENT ESTIMATION

Contact models (Section II-C) are a key component of differentiable simulators, however, existing implementations only rely on simplified approximations of real contact dynamics through compliant models or optimization-based models for ease of implementation and differentiation. It is currently not clear how these simplifications affect the differentiable simulation performance, both in terms of the quality of generated gradients and how well the simulated behavior translates to the real world. This motivates further research to investigate improving the gradients available through proposing better contact models [90]. Potential alternative approaches to obtaining more accurate or useful gradients include methods such as interpolating between zeroth and first-order gradients [127], or using a perturbed [42] or averaged gradient over a probabilistic distribution about a point [75], [126].

2) LEVERAGING GRADIENT INFORMATION

Effectively leveraging the available gradients is another area of exploration. Due to the limitations of current differentiable simulators and the gradients they provide, methods that augment local gradient-based optimizations with global search strategies such as Bayesian optimization or evolutionary algorithms is a promising path towards finding high performing solutions of nonconvex optimization problems. An example of this is [124] where three levels of optimization are used to find high performing parameterized policies. Similarly, [45] utilizes basin-hopping, which combines a local search (*e.g.*, gradient descent) with random perturbations to escape local minima. Policy optimization as an application area is comparatively under-researched and due to the slower than real-time speed of trajectory optimization using a differentiable simulator [126], policy optimization will remain an important area of research in the future. Being able to utilize the gradients available from the simulator to find high-performing policies parameterized using a neural network, similar to [43] will become increasingly important in future as it will allow for online, reactive policies to be developed. Finally, exploring the use of different loss functions or novel parametrisation methods [117] to better leverage gradient information and enhance computational efficiency are also promising directions worth investigating.

3) LONG HORIZON TASKS

Optimization over long horizons is an open problem across research fields with different communities experiencing unique aspects of this problem and proposing solutions accordingly, *e.g.* reinforcement learning [128], motion planning in robotics [129] and recurrent neural networks [130]. Differentiable simulators share some common challenges associated with long time horizons, *e.g.* vanishing and exploding gradients [43], but also have some unique problems. For example, the large increase in memory and computational cost when calculating gradients over extended simulations. To assist the associated memory cost, checkpointing [39] has been proposed. Also, as the gradients from a differentiable simulator only provide local information, this often causes the optimization of long-horizon trajectories to find local optima [131]. DiffSkill [113] overcomes local optima by breaking the long-horizon problem down by using intermediary goals and skill abstraction. Although, some aspects of long-horizon optimization problems are shared between fields, we argue that some aspects of this problem will be unique to users of differentiable simulators. Consequently, we foresee this research direction gaining prominence in the future, particularly as the complexity and duration of simulated tasks continue to increase.

4) DIFFERENTIABLE SENSOR SIMULATION

The advancement of differentiable simulators can be enhanced by the incorporation of differentiable sensor models like those offered in Sundaresan et al. [106], Xu et al. [123] and Jatavallabhula et al. [8]. These models facilitate a robust real-to-sim and sim-to-real workflow, crucial for tasks like environmental perception and interaction in robotics. Incorporating approaches such as differentiable rendering of images and point clouds, through frameworks like PyTorch3D [132], not only augments the realism and practicality of simulations but also preserves their differentiable nature. This trend paves the way for further exploration into diverse sensor models and the integration of multiple differentiable sensors into simulators, mirroring real-world sensor capabilities.

5) ONLINE APPLICATIONS

Optimizing actions online in real-time using gradients from differentiable simulators is an appealing concept. However, the present-day differentiable simulators operate at or below real-time speeds [126], limiting their application within online optimization frameworks. This research direction, explored by Chen et al. [74], holds immense potential, though the real-world tasks demonstrated so far have been constrained to relatively simple operations due to the current limitations of simulators.

6) REAL-TO-SIM TRANSFER LEARNING

The domain of real-to-sim transfer learning, where comprehensive environmental models are reconstructed within simulators to facilitate downstream tasks like planning and trajectory optimization, is another application for differentiable simulators we expect to see grow in the future [59], [106]. This direction, encompassing a spectrum of modeling challenges beyond those addressed in system identification (Section IV-A), has witnessed notable advancements in representing environmental dynamics within simulators while leveraging their differentiable properties [111], [126].

7) CHALLENGES FOR WIDESPREAD ADAPTATION

Differentiable simulation offers several advantages over current simulation suites. Open-source simulators like Brax [35], Warp [83], and TDS [27] support parallel simulation, leveraging GPU capabilities for simultaneous environments, which has been utilized in the literature for rapid model training in tasks like imitation learning [116]. Furthermore, most differenitable simulators seamlessly integrate into common workflows such as PyTorch or JAX, facilitating their incorporation into specific environments. However, challenges persist, including limited sensing capabilities, perceived complexity with a steeper learning curve for APIs, and comparatively fewer features than established simulation software. Moreover, the necessity of gradients is not universally established, especially in reinforcement learning applications, necessitating further research. Additionally, optimization techniques tailored to the nuanced landscapes of differentiable simulation require development to fully realize its potential.

VI. CONCLUSION

This review has highlighted the significant advancements and diverse applications of differentiable simulators, underscoring their transformative role in computational physics and machine learning. With the aim of answering the questions: what are differentiable simulators, how do they work and how have they been used in research thus far, this review has explored the field, covering the foundations and the core components of differentiable simulators. Looking to the future of differentiable simulators, it is evident that the continued development and refinement of differentiable simulators will be pivotal in advancing fields that rely on physically informed gradients such as robotics, control, machine learning and others.

REFERENCES

[1] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, vol. 9, pp. 51416–51431, 2021.

[2] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, "Orbit: A unified simulation framework for interactive robot learning environments," *IEEE Robot. Autom. Lett.*, vol. 8, no. 6, pp. 3740–3747, Jun. 2023.

[3] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," 2019, *arXiv:1910.07113*.

[4] J. Degrave, M. Hermans, J. Dambre, and F. Wyffels, "A differentiable physics engine for deep learning in robotics," *Frontiers Neurorobotics*, vol. 13, Mar. 2019.

[5] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "DiffTaichi: Differentiable programming for physical simulation," in *Proc. ICLR*, 2020.

[6] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, "ADD: Analytically differentiable dynamics for multi-body systems with frictional contact," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1–15, Nov. 2020.

[7] T. Du, K. Wu, P. Ma, S. Wah, A. Spielberg, D. Rus, and W. Matusik, "DiffPD: Differentiable projective dynamics," *ACM Trans. Graph.*, vol. 41, no. 2, pp. 1–21, Apr. 2022.

[8] K. M. Jatavallabhula, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Levesque, K. Xie, K. Erleben, L. Paull, F. Shkurti, D. Nowrouzezahrai, and S. Fidler, "gradSim: Differentiable simulation for system identification and visuomotor control," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[9] Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan, "Plasticinelab: A soft-body manipulation benchmark with differentiable physics," in *Proc. Int. Conf. Learn. Represent.*, 2021.

[10] E. Heiden, M. Macklin, Y. S. Narang, D. Fox, A. Garg, and F. Ramos, "DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting," *Proc. Robot., Sci. Syst.*, Jul. 2021.

[11] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and P. Agrawal, "An end-to-end differentiable framework for contact-aware robot design," in *Proc. Robot., Sci. Syst.*, Jul. 2021.

[12] S. Li, Z. Huang, T. Chen, T. Du, H. Su, J. B. Tenenbaum, and C. Gan, "DexDeform: Dexterous deformable object manipulation with human demonstrations and differentiable physics," in *Proc. 11th Int. Conf. Learn. Represent.*, 2023.

[13] Y. Li, T. Du, K. Wu, J. Xu, and W. Matusik, "DiffCloth: Differentiable cloth simulation with dry frictional contact," *ACM Trans. Graph.*, vol. 42, no. 1, pp. 1–20, Feb. 2023.

[14] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik, "ChainQueen: A real-time differentiable physical simulator for soft robotics," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 6265–6271.

[15] M. Lutter, J. Silberbauer, J. Watson, and J. Peters, "Differentiable physics models for real-world offline model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 4163–4170.

[16] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and k. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., Red Hook, NY, USA: Curran Associates, 2016, pp. 4509–4517.

[17] M. Chang, T. Ullman, A. Torralba, and J. Tenenbaum, "A compositional object-based approach to learning physical dynamics," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[18] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. F. Fei-Fei, J. Tenenbaum, and D. L. Yamins, "Flexible neural representation for physics prediction," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Red Hook, NY, USA: Curran Associates, 2018, pp. 8813–8824.

[19] C. Schenck and D. Fox, "SPNets: Differentiable fluid dynamics for deep neural networks," in *Proc. 2nd Conf. Robot Learn.*, vol. 87, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., 2018, pp. 317–335.

[20] H. Shi, Q. Yao, and Y. Li, "Learning to simulate crowd trajectories with graph networks," in *Proc. ACM Web Conf.*, Apr. 2023, pp. 8459–8468.

[21] Y. Li, J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake, "Propagation networks for model-based control under partial observation," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 1205–1211.

[22] F. D. A. Belbute-Peres, T. Economon, and Z. Kolter, "Combining differentiable PDE solvers and graph neural networks for fluid flow prediction," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 2402–2411.

[23] B. Ummenhofer, L. Prantl, N. Thuerey, and V. Koltun, "Lagrangian fluid simulation with continuous convolutions," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[24] N. Wandel, M. Weinmann, and R. Klein, "Learning incompressible fluid dynamics from scratch—Towards fast, differentiable fluid models that generalize," in *Proc. Int. Conf. Learn. Represent.*, 2021.

[25] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 5026–5033.

[26] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.

[27] E. Heiden, D. Millard, H. Zhang, and G. S. Sukhatme, "Interactive differentiable simulation," 2019, *arXiv:1905.10706*.

[28] E. Heiden, D. Millard, and G. S. Sukhatme, "Real2sim transfer using differentiable physics," in *Proc. R, SS Workshop Closing Reality Gap Sim2real Transf. Robotic Manipulation*, 2019.

[29] J. Liang, M. Lin, and V. Koltun, "Differentiable cloth simulation for inverse problems," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[30] P. Holl, V. Koltun, K. Um, and N. Thuerey, "*phiflow*: A differentiable pde solving framework for deep learning via physical simulations," in *Proc. NeurIPS Workshop*, vol. 2, 2020, pp. 1–5.

[31] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin, "Scalable differentiable physics for learning and control," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020.

[32] C. Song and A. Boularias, "Learning to slide unknown objects with differentiable physics simulations," in *Proc. Robot., Sci. Syst. XVI*, Jul. 2020.

[33] M. Ding, Z. Chen, T. Du, P. Luo, J. Tenenbaum, and C. Gan, "Dynamic visual reasoning by learning differentiable physics models from video and language," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 887–899.

[34] T. Du, J. Hughes, S. Wah, W. Matusik, and D. Rus, "Underwater soft robot modeling and control with differentiable simulation," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4994–5001, Jul. 2021.

[35] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax—A differentiable physics engine for large scale rigid body simulation," in *Proc. 35th Conf. Neural Inf. Process. Syst. Datasets Benchmarks Track (Round 1)*, 2021. [Online]. Available: https://openreview.net/forum?id=VdvDlnnjzIN

[36] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, "NeuralSim: Augmenting differentiable simulators with neural networks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 9474–9481.

[37] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier, "Differentiable simulation for physical system identification," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 3413–3420, Apr. 2021.

[38] M. A. Z. Mora, M. Peychev, S. Ha, M. Vechev, and S. Coros, "PODS: Policy optimization via differentiable simulation," in *Proc. 38th Int. Conf. Mach. Learn.*, vol. 139, M. Meila and T. Zhang, Eds., 2021, pp. 7805–7817.

[39] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin, "Efficient differentiable simulation of articulated bodies," in *Proc. 38th Int. Conf. Mach. Learn.*, vol. 139, 2021, pp. 8661–8671.

[40] A. Scibior, V. Lioutas, D. Reda, P. Bateni, and F. Wood, "Imagining the road ahead: Multi-agent trajectory prediction via differentiable simulation," in *Proc. IEEE Int. Intell. Transp. Syst. Conf. (ITSC)*, Sep. 2021, pp. 720–725.

[41] K. Wang, M. Aanjaneya, and K. Bekris, "Sim2Sim evaluation of a novel data-efficient differentiable physics engine for tensegrity robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 1694–1701.

[42] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, "Fast and feature-complete differentiable physics for articulated rigid bodies with contact," in *Proc. Robot., Sci. Syst.*, 2021.

[43] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin, "Accelerated policy learning with parallel differentiable simulation," in *Proc. Int. Conf. Learn. Represent.*, 2021.

[44] Y. D. Zhong, B. Dey, and A. Chakraborty, "Extending Lagrangian and Hamiltonian neural networks with differentiable contact models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., Red Hook, NY, USA: Curran Associates, 2021, pp. 21910–21922.

[45] E. Gärtner, M. Andriluka, E. Coumans, and C. Sminchisescu, "Differentiable dynamics for articulated 3D human motion reconstruction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 13180–13190.

[46] D. Gong, Z. Zhu, A. J. Bulpitt, and H. Wang, "Fine-grained differentiable physics: A yarn-level model for fabrics," in *Proc. Int. Conf. Learn. Represent.*, 2022.

[47] E. Granados, A. Boularias, K. Bekris, and M. Aanjaneya, "Model identification and control of a low-cost mobile robot with omnidirectional wheels using differentiable physics," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 1358–1364.

[48] T. A. Howell, S. Le Cleac'h, J. Brüdigam, J. Zico Kolter, M. Schwager, and Z. Manchester, "Dojo: A differentiable physics engine for robotics," 2022, *arXiv:2203.00806*.

[49] E. Nava, J. Z. Zhang, M. Y. Michelis, T. Du, P. Ma, B. F. Grewe, W. Matusik, and R. K. Katzschmann, "Fast aquatic swimmer optimization with differentiable projective dynamics and neural network hydrodynamic models," in *Proc. 39th Int. Conf. Mach. Learn.*, vol. 162, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., 2022, pp. 16413–16427.

[50] V. Petrík, M. N. Qureshi, J. Sivic, and M. Tapaswi, "Learning object manipulation skills from video via approximate differentiable physics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 7375–7382.

[51] D. Turpin, L. Wang, E. Heiden, Y.-C. Chen, M. Macklin, S. Tsogkas, S. Dickinson, and A. Garg, "Grasp'd: Differentiable contact-rich grasp synthesis for multi-fingered hands," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2022, pp. 201–221.

[52] Y. Wang, J. Verheul, S.-H. Yeo, N. K. Kalantari, and S. Sueda, "Differentiable simulation of inertial musculotendons," *ACM Trans. Graph.*, vol. 41, no. 6, pp. 1–11, Nov. 2022.

[53] K. Wang, M. Aanjaneya, and K. Bekris, "A recurrent differentiable engine for modeling tensegrity robots trainable with low-frequency data," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 3230–3237.

[54] A. Zhao, T. Du, J. Xu, J. Hughes, J. Salazar, P. Ma, W. Wang, D. Rus, and W. Matusik, "Automatic co-design of aerial robots using a graph grammar," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 11260–11267.

[55] A. Zhao, J. Xu, J. Salazar, W. Wang, P. Ma, D. Rus, and W. Matusik, "Graph grammar-based automatic design for heterogeneous fleets of underwater robots," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 3143–3149.

[56] T. Stuyck and H.-Y. Chen, "DiffXPBD: Differentiable position-based simulation of compliant constraint dynamics," *Proc. ACM Comput. Graph. Interact. Techn.*, vol. 6, no. 3, pp. 1–14, Aug. 2023.

[57] D. A. Bezgin, A. B. Buhendwa, and N. A. Adams, "JAX-fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows," *Comput. Phys. Commun.*, vol. 282, Jan. 2023, Art. no. 108527. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465522002466

[58] S. Chen, Y. Xu, C. Yu, L. Li, X. Ma, Z. Xu, and D. Hsu, "Daxbench: Benchmarking deformable object manipulation with differentiable physics," in *Proc. ICLR*, 2023.

[59] S. Le Cleac'h, H.-X. Yu, M. Guo, T. Howell, R. Gao, J. Wu, Z. Manchester, and M. Schwager, "Differentiable physics simulation of dynamics-augmented neural objects," *IEEE Robot. Autom. Lett.*, vol. 8, no. 5, pp. 2780–2787, May 2023.

[60] F. Liu, E. Su, J. Lu, M. Li, and M. C. Yip, "Robotic manipulation of deformable rope-like objects using differentiable compliant position-based dynamics," *IEEE Robot. Autom. Lett.*, vol. 8, no. 7, pp. 3964–3971, Jul. 2023.

[61] A. Spielberg, T. Du, Y. Hu, D. Rus, and W. Matusik, "Advanced soft robot modeling in ChainQueen," *Robotica*, vol. 41, no. 1, pp. 74–104, Jan. 2023.

[62] D. Turpin, T. Zhong, S. Zhang, G. Zhu, E. Heiden, M. Macklin, S. Tsogkas, S. Dickinson, and A. Garg, "Fast-Grasp'D: Dexterous multi-finger grasp generation through differentiable simulation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 8082–8089.

[63] T.-H. Wang, P. Ma, A. E. Spielberg, Z. Xian, H. Zhang, J. B. Tenenbaum, D. Rus, and C. Gan, "SoftZoo: A soft robot co-design benchmark for locomotion in diverse environments," in *Proc. 11th Int. Conf. Learn. Represent.*, 2023.

[64] N. Wiedemann, V. Wüest, A. Loquercio, M. Müller, D. Floreano, and D. Scaramuzza, "Training efficient controllers via analytic policy gradient," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 1349–1356.

[65] Z. Xian, B. Zhu, Z. Xu, H.-Y. Tung, A. Torralba, K. Fragkiadaki, and C. Gan, "FluidLab: A differentiable environment for benchmarking complex fluid manipulation," in *Proc. 11th Int. Conf. Learn. Represent.*, 2023.

[66] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.*, vol. 18, pp. 1–43, Apr. 2018.

[67] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *Proc. NIPS*, 2017.

[68] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: Composable transformations of Python+NumPy programs," 2018.

[69] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proc. Intl. Conf. Mach. Learn.*, 2017, pp. 136–145.

[70] S. Zimmermann, R. Poranne, and S. Coros, "Optimal control via second order sensitivity analysis," 2019, *arXiv:1905.08534*.

[71] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly, *Projective Dynamics: Fusing Constraint Projections for Fast Simulation*, 1st ed., New York, NY, USA: Association for Computing Machinery, 2014.

[72] M. Macklin, M. Müller, and N. Chentanez, "XPBD: Position-based simulation of compliant constrained dynamics," in *Proc. 9th Int. Conf. Motion Games*. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 49–54.

[73] R. Featherstone, *Rigid Body Dynamics Algorithms*. Cham, Switzerland: Springer, 2014.

[74] S. Chen, K. Werling, A. Wu, and C. K. Liu, "Real-time model predictive control and system identification using differentiable simulation," *IEEE Robot. Autom. Lett.*, vol. 8, no. 1, pp. 312–319, Jan. 2023.

[75] Q. Le Lidec, L. Montaut, C. Schmid, I. Laptev, and J. Carpentier, "Augmenting differentiable physics with randomized smoothing," in *Proc. RSS Robot. Sci. Syst., Workshop Differentiable Simulation Robot.*, New York, NY, USA, Jun. 2022.

[76] D. Morin, *Introduction to Classical Mechanics: With Problems Solutions*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[77] M. K. Jawed, A. Novelia, and O. M. O'Reilly, *A Primer Kinematics Discrete Elastic Rods*. Cham, Switzerland: Springer, 2018.

[78] Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang, "A moving least squares material point method with displacement discontinuity and two-way rigid body coupling," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–14, Jul. 2018.

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

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

[81] E. Sifakis and J. Barbic, "FEM simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction," in *Proc. ACM SIGGRAPH Courses*. New York, NY, USA: Association for Computing Machinery, Aug. 2012.

[82] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *J. Vis. Commun. Image Represent.*, vol. 18, pp. 109–118, Apr. 2007.

[83] M. Macklin, "Warp: A high-performance Python framework for GPU simulation and graphics," in *Proc. nVIDIA GPU Technol. Conf. (GTC)*, Mar. 2022.

[84] F. Boyer and P. Fabrie, *Mathematical Tools for the Study of the Incompressible Navier-Stokes Equations andRelated Models*, vol. 183. Cham, Switzerland: Springer, 2012.

[85] K. Um, R. Brand, Y. R. Fei, P. Holl, and N. Thuerey, "Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., Red Hook, NY, USA: Curran Associates, 2020, pp. 6111–6122.

[86] P. C. Horak and J. C. Trinkle, "On the similarities and differences among contact models in robot simulation," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 493–499, Apr. 2019.

[87] Y. D. Zhong, J. Han, and G. O. Brikis, "Differentiable physics simulations with contacts: Do they have correct gradients wrt position, velocity and control?" in *Proc. ICML 2nd AI Sci. Workshop*, 2022.

[88] Q. Le Lidec, W. Jallet, L. Montaut, I. Laptev, C. Schmid, and J. Carpentier, "Contact models in robotics: A comparative analysis," 2023, *arXiv:2304.06372*.

[89] K. Wang, W. R. Johnson, S. Lu, X. Huang, J. Booth, R. Kramer-Bottiglio, M. Aanjaneya, and K. Bekris, "Real2Sim2Real transfer for control of cable-driven robots via a differentiable physics engine," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2023, pp. 2534–2541.

[90] Y. D. Zhong, J. Han, B. Dey, and G. O. Brikis, "Improving gradient computation for differentiable physics simulation with contacts," in *Proc. Learn. Dyn. Control Conf.*, 2023, pp. 128–141.

[91] C. Deul, P. Charrier, and J. Bender, "Position-based rigid-body dynamics," *Comput. Animation Virtual Worlds*, vol. 27, no. 2, pp. 103–112, 2016.

[92] P. Volino and N. Magnenat-Thalmann, "Comparing efficiency of integration methods for cloth simulation," in *Proc. Comput. Graph. Int.*, Aug. 2001, pp. 265–272.

[93] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Proc. 25th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, 1998, pp. 43–54.

[94] Y. Fang, Y. Hu, S.-M. Hu, and C. Jiang, "A temporally adaptive material point method with regional time stepping," *Comput. Graph. Forum*, vol. 37, no. 8, pp. 195–204, 2018.

[95] M. Macklin, K. Storey, M. Lu, P. Terdiman, N. Chentanez, S. Jeschke, and M. Müller, "Small steps in physics simulation," in *Proc. 18th Annu. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, Jul. 2019, pp. 1–7.

[96] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C : The Art of Scientific Computing*, 2nd ed., Cambridge, U.K.: Cambridge Univ. Press, 1992.

[97] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. Karen Liu, "DART: Dynamic animation and robotics toolkit," *J. Open Source Softw.*, vol. 3, no. 22, p. 500, Feb. 2018, doi: 10.21105/joss.00500.

[98] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand, "Taichi: A language for high-performance computation on spatially sparse data structures," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 1–16, Nov. 2019, doi: 10.1145/3355089.3356506.

[99] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Philadelphia, PA, USA: SIAM, 2008.

[100] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon, "Differentiable rendering: A survey," 2020.

[101] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso, "SU2: An open-source suite for multiphysics simulation and design," *AIAA J.*, vol. 54, no. 3, pp. 828–846, Mar. 2016, doi: 10.2514/1.j053813.

[102] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai Gym," 2016, *arXiv:1606.01540*.

[103] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

[104] J. Larsson and P. Gustafsson, "A case study in fitting area-proportional Euler diagrams with ellipses using eulerr," in *Proc. Int. Workshop Set Vis. Reasoning*, vol. 2116 Edinburgh, U.K., Apr. 2018, pp. 84–91.

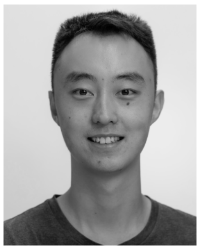
[105] E. Heiden, Z. Liu, V. Vineet, E. Coumans, and G. S. Sukhatme, "Inferring articulated rigid body dynamics from RGBD video," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 8383–8390.

[106] P. Sundaresan, R. Antonova, and J. Bohgl, "DiffCloud: Real-to-sim from point clouds with differentiable simulation and rendering of deformable objects," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 10828–10835.
[107] M. Dubied, M. Y. Michelis, A. Spielberg, and R. K. Katzschmann, "Sim-to-real for soft robots using differentiable FEM: Recipes for meshing, damping, and actuation," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 5015–5022, Apr. 2022.
[108] K. Arnavaz, M. K. Nielsen, P. G. Kry, M. Macklin, and K. Erleben, "Differentiable depth for Real2Sim calibration of soft body simulations," *Comput. Graph. Forum*, vol. 42, no. 1, pp. 277–289, Feb. 2023.
[109] J. Li, S. Bian, C. Xu, G. Liu, G. Yu, and C. Lu, "D&D: Learning human dynamics from dynamic camera," in *Proc. 17th Eur. Conf. Comput. Vis.*, Tel Aviv, Israel. Berlin, Germany: Springer, Oct. 2022, pp. 479–496.
[110] J. Lv, Y. Feng, C. Zhang, S. Zhao, L. Shao, and C. Lu, "SAM-RL: Sensing-aware model-based reinforcement learning via differentiable physics-based simulation and rendering," in *Proc. Robot., Sci. Syst. XIX*, Jul. 2023.
[111] J. Lv, Q. Yu, L. Shao, W. Liu, W. Xu, and C. Lu, "SAGCI-system: Towards sample-efficient, generalizable, compositional, and incremental robot learning," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 98–105.
[112] J. Collins, R. Brown, J. Leitner, and D. Howard, "Follow the gradient: Crossing the reality gap using differentiable physics (realitygrad)," 2021, *arXiv:2109.04674*.
[113] X. Lin, Z. Huang, Y. Li, D. Held, J. B. Tenenbaum, and C. Gan, "DiffSkill: Skill abstraction from differentiable physics for deformable object manipulations with tools," in *Proc. Int. Conf. Learn. Represent.*, 2022.
[114] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.
[115] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol Comput.*, vol. 11, no. 1, pp. 1–18, 2003.
[116] S. Chen, X. Ma, and Z. Xu, "Imitation learning as state matching via differentiable physics," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 7846–7855.
[117] M. Li, R. Antonova, D. Sadigh, and J. Bohg, "Learning tool morphology for contact-rich manipulation tasks with differentiable simulation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 1859–1865.
[118] A. Choi, R. Jing, A. P. Sabelhaus, and M. K. Jawed, "DisMech: A discrete differential geometry-based physical simulator for soft robots and structures," *IEEE Robot. Autom. Lett.*, vol. 9, no. 4, pp. 3483–3490, Apr. 2024.
[119] M. Mezghanni, T. Bodrito, M. Boulkenafed, and M. Ovsjanikov, "Physical simulation layer for accurate 3D modeling," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 13504–13513.
[120] J. R. Nieto and A. Susín, "Cage based deformations: A survey," in *Deformation Models: Tracking, Animation Applications*. Cham, Switzerland: Springer, 2012, pp. 75–99.
[121] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[122] J. Ren, C. Yu, S. Chen, X. Ma, L. Pan, and Z. Liu, "Diffmimic: Efficient motion mimicking with differentiable physics," in *Proc. 11th Int. Conf. Learn. Represent.*, 2023.
[123] J. Xu, S. Kim, T. Chen, A. R. Garcia, P. Agrawal, W. Matusik, and S. Sueda, "Efficient tactile simulation with differentiability for robotic manipulation," in *Proc. 6th Annu. Conf. Robot Learn.*, 2022, pp. 1488–1498.
[124] R. Antonova, J. Yang, K. M. Jatavallabhula, and J. Bohg, "Rethinking optimization with differentiable simulation from a global perspective," in *Proc. 6th Conf. Robot Learn.*, vol. 205, 2023, pp. 276–286.
[125] Z. Huang, L. Liang, Z. Ling, X. Li, C. Gan, and H. Su, "Variational reparametrized policy learning with differentiable physics," in *Proc. Deep Reinforcement Learn. Workshop NeurIPS*, 2022.
[126] X. Zhu, J. Ke, Z. Xu, Z. Sun, B. Bai, J. Lv, Q. Liu, Y. Zeng, Q. Ye, and C. Lu, "Diff-LfD: Contact-aware model-based learning from visual demonstration for robotic manipulation via differentiable physics-based simulation and rendering," in *Proc. 7th Annu. Conf. Robot Learn.*, 2023, pp. 499–512.
[127] H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake, "Do differentiable simulators give better policy gradients?" in *Proc. 39th Int. Conf. Mach. Learn.*, vol. 162, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., Jul. 2022, pp. 20668–20696.
[128] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Red Hook, NY, USA: Curran Associates, 2018.
[129] J. Yamada, C.-M. Hung, J. Collins, I. Havoutis, and I. Posner, "Leveraging scene embeddings for gradient-based motion planning in latent space," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 5674–5680.
[130] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1310–1318.
[131] J. Yamada, S. Zhong, J. Collins, and I. Posner, "D-Cubed: Latent diffusion trajectory optimisation for dexterous deformable manipulation," 2024, *arXiv:2403.12861*.
[132] J. Johnson, N. Ravi, J. Reizenstein, D. Novotny, S. Tulsiani, C. Lassner, and S. Branson, "Accelerating 3D deep learning with PyTorch3D," in *Proc. SIGGRAPH Asia Courses*. New York, NY, USA: Association for Computing Machinery, Nov. 2020.

RHYS NEWBURY received the B.Eng. degree (Hons.) in mechatronic engineering and the B.Sci. degree in computer science from Monash University, Australia, where he is currently pursuing the Ph.D. degree. His research interests include manipulation and using reinforcement learning.

JACK COLLINS (Member, IEEE) received the B.Eng. degree (Hons.) in mechatronic engineering and Ph.D. degree in robotics from the Queensland University of Technology, Australia, in 2017 and 2022, respectively. Since 2021, he has been a Postdoctoral Researcher with Oxford Robotics Institute, Applied AI Laboratory, University of Oxford. His research interests include simulation and the sim-to-real gap, representation learning for scene understanding and prediction, learning from demonstration, and task and motion planning for long-horizon tasks.

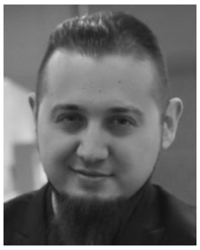
KERRY HE received the B.Eng. (Hons.) and B.Com. degrees from Monash University, during which he worked on optimal control and trajectory planning for robotic manipulators and driverless vehicles, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer System Engineering. His current research interest includes convex optimization methods for problems arising from quantum information theory.



JIAHE PAN received the B.Sc. degree in mechatronics engineering from The University of Melbourne, Australia, in 2023. He is currently a Research Assistant with the Human-Robot Interaction Group, The University of Melbourne. His current research interest includes designing adaptive autonomous robots for human–robot collaboration.

INGMAR POSNER (Member, IEEE) leads the Applied Artificial Intelligence Laboratory, University of Oxford. He is currently the Founding Director of Oxford Robotics Institute. He is an Amazon Scholar. His research aims to enable machines to robustly act and interact in the real world, with, and alongside humans. With a significant track record of contributions in machine perception and decision-making, he and his team are thinking about the next generation of robots that are flexible enough in their scene understanding, physical interaction, and skill acquisition to learn and carry out new tasks. His research is guided by a vision to create machines which constantly improve through experience. In 2014, he co-founded Oxa, a multi-award-winning provider of mobile autonomy software solutions.

DAVID HOWARD (Member, IEEE) received the B.S. and M.Sc. degrees from the University of Leeds, U.K., in 2005 and 2006, respectively, and the Ph.D. degree from the University of the West of England, U.K., in 2011. He has been with the Commonwealth Scientific and Industrial Research Organization, Brisbane, Australia, since 2013. His research interests include embodied cognition, the reality gap, and soft robotics.



AKANSEL COSGUN received the Ph.D. degree in robotics from Georgia Institute of Technology, USA, in 2016. He is currently a Senior Lecturer with Deakin University, Australia. From 2018 to 2022, he was a Research Fellow with Monash University, Australia. He conducts research in robotics, human–robot interaction, and robot learning. He has previously worked with Honda Research, Toyota Infotechnology Center, Microsoft Research, and Savioke. His research interests include mobile robots, robotic arms, and self-driving cars with an emphasis on a systems view of problems.