DiffCloth: Differentiable Cloth Simulation with Dry Frictional Contact

YIFEI LI, MIT CSAIL, USA
TAO DU, MIT CSAIL, USA
KUI WU, Tencent Lightspeed & Quantum Studios, USA
JIE XU, MIT CSAIL, USA
WOJCIECH MATUSIK, MIT CSAIL, USA

Fig. 1. We present a differentiable cloth simulator with dry frictional contact and demonstrate its efficacy in multiple downstream applications, including designing this twirl dress (10902 degrees of freedom and 125 time steps with a step size of 1/120 seconds). The goal is to optimize the material parameters of the dress so that the apex angle of the cone it forms after a twirl reaches a desired value (100 degrees in this example). Left and middle: the optimized dress before and after a twirl. Top right and bottom right: motion sequences of the twirl dress before and after material parameter optimization using our differentiable simulator. The final apex angles before and after optimization are 39.06 and 100.30 degrees, respectively.

Cloth simulation has wide applications in computer animation, garment design, and robot-assisted dressing. This work presents a differentiable cloth simulator whose additional gradient information facilitates cloth-related applications. Our differentiable simulator extends a state-of-the-art cloth simulator based on Projective Dynamics (PD) and with dry frictional contact [Ly et al. 2020]. We draw inspiration from previous work [Du et al. 2021] to propose a fast and novel method for deriving gradients in PD-based cloth simulation with dry frictional contact. Furthermore, we conduct a comprehensive analysis and evaluation of the usefulness of gradients in contact-rich cloth simulation. Finally, we demonstrate the efficacy of our simulator in a number of downstream applications, including system identification, trajectory optimization for assisted dressing, closed-loop control, inverse design, and real-to-sim transfer. We observe a substantial speedup obtained from using our gradient information in solving most of these applications.

1 INTRODUCTION

Clothing is ubiquitous in our daily lives. With the widespread appearance of clothing in the fashion industry, film industry, computer animation, and video games, simulating cloth has been an active research topic for more than two decades. Today, research advancement in cloth simulation has unlocked various applications such as virtual try-on [Guan et al. 2012], garment design [Bartle et al. 2016; Montes et al. 2020; Umetani et al. 2011; Wang 2018], fold design [Li et al. 2018c], garment grading [Brouet et al. 2012], sagging-free inversion [Ly et al. 2018], and robot-assisted dressing [Clegg et al. 2020, 2018].

Inspired by the recent development of differentiable physics simulation and its success in rigid-body systems [de Avila Belbute-Peres et al. 2018a; Geilinger et al. 2020], fluidic systems [Du et al. 2020; Hu et al. 2020], and deformable-body systems [Geilinger et al. 2020;
Hahn et al. 2019], we argue in this paper that a large number of cloth-related applications would also benefit from a high-quality differentiable cloth simulator. The critical ingredient in previous differentiable simulators is their ability to compute gradients by backpropagating any differentiable performance metrics through simulation. Such additional gradient information unlocks gradient-based continuous optimization methods, which often bring a substantial speedup compared with their gradient-free counterparts in downstream applications.

Compared with the recent active development of differentiable simulators for rigid-body and soft-body dynamics, research work about differentiable cloth simulation is still relatively sparse. Indeed, cloth simulation introduces unique challenges from its co-dimensional dynamics and, in particular, rich contact events. While many differentiable simulators have provided solutions to derive gradients for contact models of varying complexity, their techniques typically do not expect contact to be as frequent as in cloth simulation. Deriving gradients with frequent contact and self-collisions in cloth simulation is not fully resolved even in the state-of-the-art differentiable cloth simulators [Liang et al. 2019], and our work attempts to fill this gap.

In this work, we present a differentiable cloth simulator with extra care of its contact model. We base our method on the state-of-the-art cloth simulator from Ly et al. [2020] and employ its Projective Dynamics (PD) simulation method and dry frictional contact described by the Signorini-Coulomb law. Therefore, our differentiable cloth simulator inherits both the speedup from Projective Dynamics and the physical accuracy from the dry frictional contact model. Unlike previous papers relying on automatic differentiation tools to derive gradients, we present an iterative solver modified from Du et al. [2021] to accommodate the dry frictional contact model. We show that the modified iterative solver leads to a substantial speedup over a standard linear solver in gradient computation.

To have well-defined gradients, a differentiable simulator expects all quantities computed in simulation to be sufficiently smooth. However, the non-smooth position and force changes from large numbers of contact events question this fundamental assumption. To fully understand the usefulness of a differentiable simulator in contact-rich environments, our work conducts comprehensive evaluation and analysis of the behavior of our differentiable cloth simulation with a varying number of contact events. While previous papers have provided similar discussions, they primarily focus on understanding penalty-based contact models [Geilinger et al. 2020] or discussing collisions in rigid-body dynamics [Werling et al. 2021] where contact events are not as frequent as in cloth simulation. To our best knowledge, our work is the first to present such evaluation and discussion of the usefulness of gradients in a differentiable cloth simulator with dry frictional contact.

We demonstrate the efficacy of our simulator in various applications, including system identification of frictional coefficients in cloth simulation, inverse garment design for computer animation, and motion planning of robotic manipulators in robot-assisted dressing. Many of these applications would either be impossible or require a much longer time to solve with existing methods. With the extra gradient information from our differentiable simulator, we unlock gradient-based optimizers to solve these problems with a much higher sample efficiency than traditional gradient-free methods.

To summarize, our work contributes the following:

- We present a novel differentiable cloth simulator with dry frictional contact and an iterative solver for speeding up its gradient computation.
- We evaluate the source of non-differentiability in the dry frictional contact model and discuss the usefulness of gradients in differentiable, contact-rich cloth simulation.
- We show the efficacy of our simulator in various applications, including system identification, trajectory optimization for assisted dressing, closed-loop control, inverse design, and real-to-sim transfer.

2 RELATED WORK

Our work is closely related to cloth simulation and its applications in computer graphics. It is also relevant to the more recent differentiable simulation methods developed in the graphics and machine learning communities.

Cloth simulation. Physics-based cloth simulation has been a popular topic in the graphics community for decades since clothing has been widely used in our daily lives. The implicit Euler integration is used to simulate cloth robustly with large time steps [Baraff and Witkin 1998; Terzopoulos et al. 1987] while introducing excessive numerical damping. To alleviate this problem, implicit and explicit methods (IMEX) [Bridson et al. 2003; Stern and Grinspun 2009] explicitly integrate elastic forces and implicitly integrate damping forces. Researchers have also introduced other variational integrators, e.g., BDF2 [Choi and Ko 2002] and Symplectic integrator [Stern and Desbrun 2006], to conserve the total energy of the system.

For cloth modeled as a mass-spring system, Liu et al. [2013] treat the implicit Euler integration as an energy minimization problem. The global linear system then remains constant on run-time, and each spring constraint can be solved separately in a local step. That global/local solver idea is generalized as Projective Dynamics (PD) [Bouaziz et al. 2014], which supports material models whose elastic energy has a specific quadratic form. Projective Dynamic is further extended to support more general materials [Liu et al. 2017; Overby et al. 2017]. Though a local step in PD can be processed in parallel, the global step still needs to maintain a large pre-factorized sparse matrix and do back substitutions in each step. Another relevant idea is Position-Based Dynamics (PBD) [Macklin et al. 2016; Müller et al. 2007], which iteratively projects each constraint in a non-linear Gauss-Seidel-like fashion, leading to highly parallelizable computation on GPUs.

Both PD and PBD have led to a few follow-up works on more advanced speedup techniques. Komaritzan and Botsch [2018] present techniques to speed up PD with contact for physics-based character skinning. For PD and PBD in cloth simulation, Wang [2015] and a follow-up work [Wang and Yang 2016] propose a Chebyshev acceleration technique that can be applied to both PD and PBD to speed up convergence. Fratarcangeli et al. [2016] also introduce a parallel randomized Gauss-Seidel method that re-organizes the unknowns of the sparse linear system into a few independent blocks, which
can be solved in parallel with a single Gauss-Seidel step. Recently, the computation of integration is further accelerated by the multigrid method, e.g., geometric multigrid scheme [Wang et al. 2018], algebraic multigrid scheme [Tamstorf et al. 2015], and Galerkin multigrid scheme [Xian et al. 2019].

Lastly, our work is also relevant to previous attempts to matching cloth simulation with real fabric behaviors [Clyde et al. 2017; Miguel et al. 2012, 2013; Wang et al. 2011], which is typically done by fitting constitutive material models with real material properties and can benefit from extra gradient information from a differentiable simulator [Hahn et al. 2019].

**Cloth contact and friction.** Contact and friction are key ingredients in modern cloth simulation. Provot [1997] proposes a method called “impact zones” to collect the nodes involved in multiple collisions which is still widely used in modern cloth simulators [Li et al. 2020b; 2021; Mistry et al. 2010]. Inverse dynamics for high-DoF systems degrees of freedom (DoFs) [Dario Bellicoso et al. 2016; Kang et al. 2018] are important ingredients in the motion of the incriminated vertices. Bridson et al. [2002] introduce a hybrid method to combine the idea of applying impulse impulses, frictions, and impact zones to handle cloth collision robustly, which is still widely used in modern cloth simulators [Li et al. 2020b; Narain et al. 2012]. To model friction fully implicitly, Brown et al. [2018] treat friction as an additional dissipative term in optimization. Regarding contact handling, repulsive forces or penalty methods have been widely used [Bouaziz et al. 2014; Geilinger et al. 2020; Macklin et al. 2020; Wang 2015] since they are easy to implement. However, these methods often need high stiffness in the penalty energy, leading to disturbing jittering artifacts that demand careful tuning. Recently, Li et al. [2020a, 2021] define a smooth dissipative potential for friction using barrier methods. Their method can guarantee interpenetration-free states after each time step without the need for parameter tuning.

Unlike penalty-based methods, constraint-based collision handling methods formulate contact as constraints in the physics system. Otaduy et al. [2009] first formulate cloth contacts as a sparse linear complementarity problem (LCP). Their key idea is to interleave frictional contact iterations with normal contact iterations. Instead of using a pyramid approximation of the Coulomb friction cone [Otaduy et al. 2009], Li et al. [2018b] rely upon the exact Coulomb friction cone and adaptively refine nodes to ensure an accurate treatment of frictional contact. Although constraint-based methods typically ensure the physics-based constraints characterizing contact and friction are satisfied after time integration, their computation cost is typically much more expensive than a penalty-based method. Recently, Ly et al. [2020] propose an efficient algorithm to incorporate frictional contact into Projective Dynamics so that non-penetration and Coulomb constraints are satisfied simultaneously in a semi-implicit way.

**Inverse dynamics.** Inverse dynamics have been studied in robotics for decades to reconstruct internal forces or torques from the observations of robotic systems. However, existing methods usually focus on rigid-body systems only, which have less than a hundred degrees of freedom (DoFs) [Dario Bellicoso et al. 2016; Kang et al. 2021; Mistry et al. 2010]. Inverse dynamics for high-DoF systems like soft bodies, fluids, and cloth are still under exploration due to the lack of high-quality numerical solutions in robotics for both simulation and differentiation. One noticeable distinction between inverse dynamics and differentiable simulation is that differentiable simulation computes additional gradients for initial states, system parameters, and design parameters. Therefore, differentiable simulation enables more applications like system identification, inverse design, and real-to-sim matching that traditional inverse dynamics typically do not consider.

**Differentiable simulation.** Differentiable simulation is a relatively recent concept explored in the graphics and machine learning community, but its original idea can be traced back to much earlier works in graphics decades ago. Perhaps one of the earliest such papers is Witkin and Kass [1988], which shows optimizing simulation to minimize an objective. Despite the recent advances in differentiable simulators in rigid-body dynamics [de Avila Belbute-Peres et al. 2018b; Degrave et al. 2019; Popović et al. 2003; Qiao et al. 2020; Toussaint et al. 2018; Xu et al. 2021], soft-body dynamics [Du et al. 2021; Geilinger et al. 2020; Hahn et al. 2019; Hu et al. 2020, 2019], and fluid dynamics [Holl et al. 2020; McNamara et al. 2004; Schenck and Fox 2018; Treuille et al. 2003; Wojtan et al. 2006], differentiable cloth simulation still lacks a good solution. One natural idea is to use particle-based strategies that approximate a nodal system of cloth with graph neural networks [Li et al. 2019; Sanchez-Gonzalez et al. 2020]. Although the neural networks are naturally differentiable, the physical accuracy is hard to guarantee.

To accurately predict the behavior of real-world objects, recent papers [Geilinger et al. 2020; Hahn et al. 2019] present differentiable soft-body systems and mass-spring systems with implicit Euler time integration and penalty-based contacts. Although a few recent papers [Li et al. 2020a, 2021; Macklin et al. 2020] have introduced differentiable contact handling methods, none of them show a clothing example using the differentiability besides demonstrating the differentiability of their methods in theory. Liang et al. [2019] are the first to introduce a fully functional differentiable cloth simulation with contact, friction, and self-collision. Instead of constructing a static LCP problem, they develop a quadratic programming (QP) problem to minimize the change between the collision-free state and the original mesh state. Murthy et al. [2021] also present a differentiable cloth simulator with penalty-based frictional contact in their fully differentiable simulation and rendering pipeline. In this paper, we build upon the differentiable PD framework [Du et al. 2021] to simulate cloth dynamics with dry frictional contact [Ly et al. 2020] and augment it with gradient computation.

A significant challenge in differentiable simulation is the presence of non-smooth contact events, where non-smoothness can arise from discontinuous contact shapes [Popović et al. 2000], impulsive forces [Hu et al. 2019], or branches in contact laws [Ly et al. 2020], to name a few. Many existing differentiable simulators assume such non-smoothness does not affect the usability of gradients. Indeed, they present empirical results that observe the benefits of using gradients in downstream applications [de Avila Belbute-Peres et al. 2018a; Du et al. 2021; Geilinger et al. 2020; Liang et al. 2019]. While our work also observes the advantages of gradients over gradient-free methods in several applications, we take one step further by analyzing the source of non-smoothness and discontinuities in contact-rich cloth simulation. We hope that our
experience can serve as useful heuristics for applying differentiable simulation methods in the future.

3 BACKGROUND

In this section, we briefly review the Projective Dynamics cloth simulation method with dry frictional contact described in Ly et al. [2020], on which our differentiable cloth simulator is based. Our core contribution lies in the development of gradient computation in this PD framework with dry frictional contact, which we detail in the next section.

3.1 Projective Dynamics Method

Implicit time integration. We model cloth as a nodal system with m 3D nodes. Let \( x(t) \) and \( v(t) \) be two vector functions from \( \mathbb{R}^+ \) to \( \mathbb{R}^{3m} \) indicating the positions and velocities of all nodes at time \( t \). We consider the standard implicit time-stepping scheme discretizing \( x(t) \) and \( v(t) \) as follows:

\[
\begin{align*}
  x_{n+1} &= x_n + hv_{n+1}, \\
  v_{n+1} &= v_n + hM^{-1}[f_{\text{int}}(x_{n+1}) + f_{\text{ext}}],
\end{align*}
\]

where \( h \) is the time step size, \( M \) a positive diagonal mass matrix of size \( 3m \times 3m \), and \( f_{\text{int}} \) and \( f_{\text{ext}} \) the internal and external forces at each node stacked as two \( 3m \)-dimensional vectors, respectively. Note that we assume for now \( f_{\text{int}} \) and \( f_{\text{ext}} \) do not contain contact forces, which will be discussed separately in Sec. 3.2. We use the cloth material model described in Bouaziz et al. [2014] to define \( f_{\text{int}} \).

The implicit time-stepping scheme connects the state of the nodal system \( (x_n, v_n) \) at time \( t_n \) to \( (x_{n+1}, v_{n+1}) \) at new time \( t_{n+1} = t_n + h \).

Optimization view. As discussed by Martin et al. [2011], the implicit time-stepping scheme can be rephrased as finding a saddle point of the following energy minimization problem:

\[
\min_{x_{n+1}} \frac{1}{2h^2}(x_{n+1} - y)^{\top}M(x_{n+1} - y) + W(x_{n+1}),
\]

where \( y = x_n + hv_n + h^2M^{-1}f_{\text{ext}} \) is a constant vector that can be precomputed at the beginning of the current time step. The potential energy \( W \) defines the internal force \( f_{\text{int}} \) by its spatial gradients: \( f_{\text{int}} = -\nabla W(x_{n+1}) \). It is easy to verify that setting the gradient of the objective function to zero leads to a system of equations identical to Eqsns. (1) and (2).

Local and global solvers in PD. The key assumptions in PD is that the internal energy \( W \) can be written as a sum of quadratic forms [Bouaziz et al. 2014; Liu et al. 2017]:

\[
\begin{align*}
  W_i(x) &= \min_{p_i \in M_1} \frac{w_i}{2} ||A_i x - p_i||_2^2, \\
  W(x) &= \sum_i W_i(x),
\end{align*}
\]

Here, each energy \( W_i \) projects \( A_i x \), a linear transformation of \( x \), to its closest point in the set \( M_1 \) and scales its squared distance by a prespecified stiffness \( w_i \). Both \( A_i \) and \( M_1 \) are predefined and independent of \( x \). Here, \( M_1 \) is the constraint manifold, and \( p_i \) is the auxiliary projection variable as defined in Bouaziz et al. [2014].

With such an assumption on \( W \), PD proposes a local-global solver to minimize a surrogate objective function:

\[
\tilde{g}(x_{n+1}, p) = \frac{1}{2h^2}(x_{n+1} - y)^{\top}M(x_{n+1} - y) + \sum_i \tilde{W}_i(x_{n+1}, p_i).
\]

where \( p \) stacks up all \( p_i \) from each \( W_i \). In the local step, PD fixes \( x_{n+1} \) and projects each \( p_i \) to its corresponding \( M_1 \) by solving Eqn. (4). Such a local step can be done in parallel for each \( \tilde{W}_i \). In the global step, PD fixes \( p \) and minimizes \( \tilde{g} \) as a function of \( x_{n+1} \), which turns out to have a closed-form solution:

\[
(M + h^2 \sum_i w_i A_i^{\top} A_i) x_{n+1} = My + h^2 \sum_i w_i A_i^{\top} p_i.
\]

By alternating between the local and global steps, PD monotonically decreases the surrogate energy \( \tilde{g} \) until convergence, which can be shown to agree with the saddle point of \( g \) [Liu et al. 2017]. The source of efficiency in PD comes from the observation that \( P \) is a constant matrix that can be prefactorized at the beginning of the simulation, leading to an efficient global step requiring back-substitution only.

3.2 Dry Frictional Contact in Projective Dynamics

Signorini-Coulomb law. Ly et al. [2020] augment the standard PD framework described above with non-penetration collision and Coulomb friction by assuming contact applies to nodes only. At each time step, assuming a collision detection algorithm has identified a set \( I \subseteq \{1, 2, 3, \ldots, m\} \) which describes the indices of contact nodes. For each node \( j \in I \), the Signorini-Coulomb law [Brogliato 2016] requires its local force \( r_j \) and velocity \( u_j \) to satisfy one of the following three conditions:

\[
\begin{align*}
\text{Take off: } r_j = 0, u_j > 0, \\
\text{Stick: } |r_j| < \mu u_j, u_j > 0, \\
\text{Slip: } |r_j| = \mu u_j, u_j = 0, |r_j| \cdot u_j > 0, |r_j| < \mu u_j.
\end{align*}
\]

Here, \( \mu \) is the frictional coefficient, and \( r_j \) and \( u_j \) are the nodal velocity and contact force represented in the local contact frame spanned by the tangential plane and contact normal on the contact surface. The notations \( jT \) and \( jN \) represent the tangential and normal components of a 3D vector defined in the local frame. We further define \( C_j \subseteq \mathbb{R}^3 \times \mathbb{R}^3 \) as the set of valid \((r_j, u_j)\) pairs described by the conditions above, allowing us to rewrite Eqn. 8 compactly:

\[
(r_j, u_j) \in C_j.
\]

Implicit time integration with dry frictional contact. The original implicit time integration now needs to be augmented with additional constraints describing contact conditions [Ly et al. 2020]:

\[
\begin{align*}
  x_{n+1} &= x_n + hv_{n+1}, \\
  v_{n+1} &= v_n + hM^{-1}[f_{\text{int}}(x_{n+1}) + f_{\text{ext}} + \int_0^h (\mathcal{J}_n x_n, v_n) f_{\text{int}}], \\
  u_{n+1} &= \int_0^h (\mathcal{J}_n x_n, v_n), \\
  (r_{n+1}, u_{n+1}, j) &\in C_j, \forall j \in I_n.
\end{align*}
\]

Here, we have added the subscript \( n \) in the definitions of the contact set \( I \) and the contact condition \( C_j \) described above to specify the
time step they are defined from. The notation \( r_{n+1,j} \) and \( u_{n+1,j} \) select the 3D force and velocity corresponding to the \( j \)-th node from \( r_{n+1} \) and \( u_{n+1} \), respectively. The Jacobian matrix \( J_n \) of size \( 3|I_n| \times 3n \) selects global vectors defined on the contact nodes and computes their coordinates in the local contact frames. Note that our definition of \( J_n \) implies that it is computed in an explicit manner, i.e., it relies on the last state \((x_n, v_n)\) entering the \( n \)-th time step. For brevity, this background section will describe simple contact events between a node and a static plane, in which case \( J_n \) is a constant matrix that does not need to be computed from \((x_n, v_n)\). Extensions to more sophisticated contact events, e.g., self-collisions between multiple contact nodes or contact events between a node and a non-planar obstacle, can be found in Ly et al. [2020] and in our implementation and experiments.

**Projective Dynamics with dry frictional contact.** The core idea in Ly et al. [2020] is an additional local step in PD that solves each contact node independently. Noting that the contact conditions are primarily defined on nodal velocities instead of nodal positions, Ly et al. [2020] first rewrite the global step in Eqn. (7) as an equation of velocities:

\[
Pv_{n+1} = \frac{1}{h} \left[ b(p) - Px_n \right].
\]

The velocity-based PD global-local steps then alternate between this new global solver and the original local step, which is unaffected by this change of variables. By comparing Eqn. (9) and Eqn. (10), we can see that incorporating the contact force adds an additional impulse \( hJ^T_n r_{n+1} \) to the right-hand side of the global step:

\[
Pv_{n+1} = \hat{b}(p) + hJ^T_n r_{n+1}.
\]

The goal of the global solver is now to find \((v_{n+1}, r_{n+1})\) that satisfy contact conditions at each \( j \). Noting that in Eqn. (11), \( P \) is the only operator that couples unknown contact forces from different \( j \), Ly et al. [2020] propose the following iterative solver based on the decomposition of \( P \) to solve Eqn. (11):

\[
Mv^k_{n+1} = \hat{b}(p) - \left( h^2 \sum_i w_i A_i^T A_i \right) v^k_{n+1} + hJ^T_n r^k_{n+1}.
\]

where the superscripts \( k \) and \( k+1 \) indicate two consecutive iterations in the iterative solver at fixed time step \( n+1 \). After iteration \( k \), \( r^k_{n+1} \) is updated using \( v^k_{n+1} \) to enforce the Signorini-Coulomb law, which is then used to update \( v^{k+1}_{n+1} \) according to Eqn. (12). It is easy to see that the proposed iterative solver fully decouples the momentum equation on each node, allowing Ly et al. [2020] to enforce the Signorini-Coulomb law by adjusting \((v_{n+1,j}, r_{n+1,j})\) on each contact node \( j \) independently in a straightforward manner. When the iterative solver converges, Ly et al. [2020] prove that the result is indeed a solution to Eqn. (9). We refer the readers to the original paper for the details.

### 4 DIFFERENTIABLE CLOTH SIMULATION

We now describe in detail how we extend the forward simulator in Sec. 3 to build a differentiable cloth simulator. We start by differentiating implicit time integration in PD without contact, followed by explaining how contact gradients can be added to this framework. Compared with other differentiable simulators, our simulator is unique because of its treatment of the contact-rich nature of cloth simulation: many existing differentiable simulators focus on sparse contact events in rigid-body or deformable-body dynamics [Du et al. 2021; Geilinger et al. 2020; Hu et al. 2019], and the state-of-the-art differentiable cloth simulation method [Liang et al. 2019] handles rich contact events but without physics-based contact forces or friction. To our best knowledge, our work is the first to present a differentiable cloth simulator that can handle rich contact events with Coulomb’s law of friction.

#### 4.1 Gradients without Contact

**Differentiating implicit time integration.** The core step in building a differentiable simulator based on implicit time-stepping scheme is to backpropagate gradients through the implicit integration described in Eqs. (1) and (2), or equivalently, to derive the Jacobian of the output \((x_{n+1}, v_{n+1})\) with respect to the input \((x_n, v_n)\). Mature techniques such as sensitivity analysis, adjoint method, and implicit function theorem have proven to be successful in computing such gradients [Du et al. 2021; Geilinger et al. 2020; Hahn et al. 2019]. To sketch the idea, we plug \( v_{n+1} \) from Eqn. (2) into Eqn. (1):

\[
x_{n+1} = x_n + h v_n + h^2 M^{-1} f_{\text{int}}(x_{n+1}) + f_{\text{ext}}
\]

which is essentially the first-order optimality condition in Eqn. (3). Differentiating \( x_n \) on both sides gives

\[
\frac{\partial x_{n+1}}{\partial x_n} = I + h^2 M^{-1} \frac{\partial f_{\text{int}}(x_{n+1})}{\partial x_{n+1}} \frac{\partial x_{n+1}}{\partial x_n}
\]

or equivalently,

\[
\frac{\partial x_{n+1}}{\partial x_n} = \left[ I + h^2 M^{-1} \nabla \mathcal{L}(x_{n+1}) \right]^{-1} \frac{\partial \mathcal{L}}{\partial x_n}.
\]

In backpropagation, such a Jacobian matrix is coupled with the gradients of a loss function \( \mathcal{L} \) which are passed to the previous state (assuming the gradients below are both column vectors):

\[
\frac{\partial \mathcal{L}}{\partial x_n} = (\frac{\partial x_{n+1}}{\partial x_n})^T \frac{\partial \mathcal{L}}{\partial x_{n+1}}
\]

\[
= M \left[ I + h^2 \nabla^2 \mathcal{L}(x_{n+1}) \right]^{-1} \frac{\partial \mathcal{L}}{\partial x_{n+1}}.
\]

Here, we obtain the adjoint vector \( z_{n+1} \) by solving the linear system of equations with the matrix \( M + h^2 \nabla^2 \mathcal{L}(x_{n+1}) \) on the left-hand side, which avoids an explicit inversion of the large and sparse matrix. Backpropagating gradients of \( \mathcal{L} \) with respect to \( v_n \) and \( v_{n+1} \) can be
derived similarly. In fact, it solves the same linear system but with a different vector \( \frac{\partial L}{\partial \mathbf{x}_{n+1}} \) on the right-hand side.

**Differentiating with Projective Dynamics.** With assumptions on the energy form \( W \) in PD, Du et al. [2021] show that we can speed up the computation in \( \mathbf{x}_{n+1} \) by exploiting the special form of \( \nabla^2 W \). The first- and second-order derivatives in Eqn. (5) are given by the following equations [Du et al. 2021; Liu et al. 2017]:

\[
\nabla W_i(\mathbf{x}) = w_i A_i^T A_i \mathbf{x} - p_i(\mathbf{x}),
\]

\[
\nabla^2 W_i(\mathbf{x}) = w_i A_i^T A_i - w_i A_i \frac{\partial p_i}{\partial \mathbf{x}},
\]

where \( p_i(\mathbf{x}) = \arg \min_{\mathbf{p} \in M_i} W_i(\mathbf{x}, \mathbf{p}) \) is the projection of \( A_i \mathbf{x} \) onto \( M_i \) obtained in the local step. Interested readers can refer to the appendix of Liu et al. [2017] for their derivation details. We plug them to \( \mathbf{z}_{n+1} \) and rewrite the linear system as follows:

\[
(P - h^2 \sum_i w_i A_i^T \frac{\partial p_i}{\partial \mathbf{x}}(\mathbf{x}_{n+1})) \mathbf{z}_{n+1} = \frac{\partial L}{\partial \mathbf{x}_{n+1}}.
\]

Noting that a prefactorization of \( P \) exists in Projective Dynamics, Du et al. [2021] propose the following iterations to solve \( \mathbf{z}_{n+1} \):

\[
P_{n+1}^k \mathbf{P} \mathbf{z}_{n+1}^k = \Delta P \mathbf{z}_{n+1}^k + \frac{\partial L}{\partial \mathbf{x}_{n+1}}.
\]

Similar to the local-global steps in Projective Dynamics, a local step can evaluate \( \Delta P \) across each \( \nabla^2 W_i \) in parallel, and a global step, which reuses \( \mathbf{P} \), can solve \( \mathbf{z}_{n+1}^k \) with backsubstitution only. Du et al. [2021] show that such a local-global solver is empirically faster than directly solving Eqn. (16), which resembles the source of efficiency in the original PD method.

### 4.2 Gradients with Contact

While Du et al. [2021] have discussed extensively how to fuse backpropagation into the Projective Dynamics framework, its support of contact is limited to non-penetration conditions without a proper treatment on friction. On the other hand, other prior papers have provided solutions to deriving gradients with contact governed by Signorini-Coulomb law, but they focus on either small-scale problems (e.g., rigid bodies in de Avila Belbute-Peres et al. [2018a]) or sparse contact events on a static ground only [Geilinger et al. 2020]. Here, we present our differentiable cloth simulator that both inherits the speedup from PD and handles gradients with complicated contact events like self-collisions, which are overlooked in many existing differentiable simulators but common in cloth simulation.

**Differentiating implicit time integration with contact.** Consider the \( n \)-th time step in simulation with contact which takes as input \( (\mathbf{x}_n, \nu_n) \) and computes \((\mathbf{x}_{n+1}, \nu_{n+1}, \mathbf{r}_{n+1}, \mathbf{u}_{n+1})\) that satisfy Eqn. (9). In particular, for each contact node \( j \), the forward simulation identifies which one of the contact conditions in Eqn. (8) applies to \((\mathbf{r}_{n+1,j}, \mathbf{u}_{n+1,j})\). As an example, assuming a contact node \( j \) satisfies Eqn. (8c), its constraints can be summarized as follows:

\[
\| \mathbf{r}_{n+1,j} | T | - \mu \mathbf{u}_{n+1,j} | N | = 0,
\]

\[
\mathbf{u}_{n+1,j} | N | = 0,
\]

\[
(\mathbf{u}_{n+1,j} | T |) x (\mathbf{r}_{n+1,j} | T |) y - (\mathbf{u}_{n+1,j} | T |) y (\mathbf{r}_{n+1,j} | T |) x = 0,
\]

\[
\mathbf{u}_{n+1,j} | T | - \mathbf{r}_{n+1,j} | T | \leq 0.
\]

where \((\cdot)_x\) and \((\cdot)_y\) extract the \( x \) and \( y \) components of a two-dimensional vector, respectively. The other two cases of Eqn. (8) similarly enforce three equality constraints \((\mathbf{r}_{n+1,j} = 0 \text{ for taking off and } \mathbf{u}_{n+1,j} = 0 \text{ for sticking})\) and a number of inequality constraints on \((\mathbf{r}_{n+1,j}, \mathbf{u}_{n+1,j})\). If the inequality constraint is inactive, slightly perturbing the inputs to the simulator will keep \((\mathbf{r}_{n+1,j}, \mathbf{u}_{n+1,j})\) inside its interior. Therefore, we can remove the inactive inequality when deducing the gradients during backpropagation. When the inequality constraint is active, the gradients of the simulation are not well defined because it represents corner cases that can be categorized into more than one contact types. These corner cases introduce non-smoothness to the simulator, but it is worth mentioning that they do not create discontinuities, just like the standard rectifier (ReLU) activation function is still continuous despite the non-smoothness at its turning point.

After removing the inequality constraint, we further define a nonlinear vector function \( C_n^j(\mathbf{r}_{n+1,j}, \mathbf{u}_{n+1,j}) \) for the left-hand side of the three equality constraints and compactly represent the constraints as \( C_n^j(\mathbf{r}_{n+1,j}, \mathbf{u}_{n+1,j}) = 0 \). It is convenient that \( C_n^j \) for all three cases in Eqn. (8) have three dimensional outputs. We can stack \( C_n^j \) from all contact nodes \( j \in I_j \) into a nonlinear function \( C_n(\cdot, \cdot) : \mathbb{R}^{3|I_j|} \times \mathbb{R}^{3|I_j|} \rightarrow \mathbb{R}^{3|I_j|} \) (note that we only evaluate the function at valid input pairs that lie in \( C^j \), allowing us to restate Eqn. (9) as follows:

\[
\begin{bmatrix}
\mathbf{v}_{n+1} = \mathbf{v}_n + h \mathbf{M}^{-1} \left[ \mathbf{f}_\text{ext}(\mathbf{x}_n + h \mathbf{v}_{n+1}) + \mathbf{f}_\text{ext} + \int_0^1 \mathbf{J}_R \mathbf{r}_{n+1} \right], \\
\mathbf{C}_n(\mathbf{r}_{n+1}, J_0 \mathbf{v}_{n+1}) = 0.
\end{bmatrix}
\]

Note that we choose \( \mathbf{v}_{n+1} \) instead of \( \mathbf{x}_{n+1} \) as our variable in order to be consistent with the forward PD simulation with contact described by Ly et al. [2020]. Differentiating both sides of the first equation with respect to \( \mathbf{v}_n \) leads to the following result:

\[
\frac{\partial \mathbf{v}_{n+1}}{\partial \mathbf{v}_n} = I + h \mathbf{M}^{-1} [ -h \mathbf{V}^2 W(x_n + h \mathbf{v}_{n+1}) + \int_0^1 \frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{v}_n} \frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{v}_n} ] \frac{\partial \mathbf{v}_{n+1}}{\partial \mathbf{v}_n} + \frac{\partial \mathbf{v}_{n+1}}{\partial \mathbf{v}_n}
\]

\[
\frac{\partial \mathbf{v}_{n+1}}{\partial \mathbf{v}_n} = [ I + h^2 \mathbf{M}^{-1} \mathbf{V}^2 W(x_n + h \mathbf{v}_{n+1}) - h \mathbf{M}^{-1} ] \frac{\partial \mathbf{v}_{n+1}}{\partial \mathbf{v}_n} + \frac{\partial \mathbf{v}_{n+1}}{\partial \mathbf{v}_n}
\]

\[
[ \mathbf{M} + h^2 \mathbf{V}^2 W(x_n + h \mathbf{v}_{n+1}) - h \mathbf{M}^{-1} ] \frac{\partial \mathbf{v}_{n+1}}{\partial \mathbf{v}_n} = \frac{\partial \mathbf{v}_{n+1}}{\partial \mathbf{v}_n}
\]

\[
\Delta \mathbf{R}^T
\]

Comparing it with Eqn. (15), we see the matrix to be inverted now has an additional component dependent on \( \frac{\partial \mathbf{C}_n}{\partial \mathbf{v}_n} \), which we obtain from differentiating the constraint \( \mathbf{C}_n = 0 \) in Eqn. (22):

\[
\frac{\partial \mathbf{C}_n}{\partial \mathbf{r}_{n+1, j_0 \mathbf{v}_{n+1}}} \frac{\partial \mathbf{r}_{n+1, j_0 \mathbf{v}_{n+1}}}{\partial \mathbf{v}_n} + \frac{\partial \mathbf{C}_n}{\partial \mathbf{u}_n} \frac{\partial \mathbf{u}_n}{\partial \mathbf{v}_n} J_0 = 0,
\]

\[
\frac{\partial \mathbf{r}_{n+1, j_0 \mathbf{v}_{n+1}}}{\partial \mathbf{v}_n} = -\left( \frac{\partial \mathbf{C}_n}{\partial \mathbf{r}_{n+1, j_0 \mathbf{v}_{n+1}}} \right)^{-1} \frac{\partial \mathbf{C}_n}{\partial \mathbf{u}_n} \frac{\partial \mathbf{u}_n}{\partial \mathbf{v}_n} J_0.
\]
We stress that computing \( \frac{\partial C_n^{\text{diff}}}{\partial \mathbf{u}} \) and \( \frac{\partial C_n^{\text{diff}}}{\partial \mathbf{x}} \) is trivial because both partial derivatives are 3 \times 3 block-diagonal matrices. Therefore, \( \Delta \mathbf{R} \) can be parallelized among all contact nodes. Backpropagation through \( \mathbf{v}_{n+1} \) to \( \mathbf{v}_n \) can be implemented with the same adjoint method before, which we give in the equation below for completeness with the notation \( \mathbf{z}_{n+1} \) overloaded:

\[
\frac{\partial L}{\partial \mathbf{v}_n} = \mathbf{M} \left[ (\mathbf{M} + h^2 \nabla^2 \mathbf{W}(\mathbf{x} + h \mathbf{v}_{n+1}) - \Delta \mathbf{R})^{-1} \frac{\partial L}{\partial \mathbf{v}_{n+1}} \right].
\]

(28)

Speedup with Projective Dynamics. With additional information about \( \mathbf{W} \) from PD, we can rewrite the adjoint vector \( \mathbf{z}_{n+1} \) in Eqn. (28) by comparing it with Eqn. (19):

\[
(\mathbf{P} - \Delta \mathbf{P} - \Delta \mathbf{R})\mathbf{z}_{n+1} = \frac{\partial L}{\partial \mathbf{v}_{n+1}},
\]

which naturally leads to the following iterative solver:

\[
\mathbf{P} \mathbf{z}^{k+1}_{n+1} = (\Delta \mathbf{P} + \Delta \mathbf{R}) \mathbf{z}^{k}_{n+1} + \frac{\partial L}{\partial \mathbf{v}_{n+1}}.
\]

(29)

Comparing it with Eqn. (20), we see the role of \( \Delta \mathbf{R} \) is replaced with \( \Delta \mathbf{P} + \Delta \mathbf{R} \). Since we have shown that computing \( \Delta \mathbf{R} \) can be massively parallelized among contact nodes, it is suitable for contact-rich cloth simulation and preserves the efficiency of the local-global solver.

Convergence. In theory, such an iterative solver is guaranteed to converge from any initial guesses of \( \mathbf{z}_{n+1} \) when the spectral radius \( \rho([\mathbf{P}^{-1}(\Delta \mathbf{P} + \Delta \mathbf{R})]) < 1 \). Empirically, we notice in our cloth simulation that divergence is uncommon, especially when high-precision backpropagation is not required (Sec. 6). When the iterative solver fails to converge, we switch back to a direct sparse matrix solver to solve Eqn. (29).

Extensions. We deliberately skipped the full definition of \( \mathbf{J}_n \) in all equations above for a clearer presentation of the main idea behind our differentiable cloth simulation. We now elaborate on the role of \( \mathbf{J}_n \) and discuss its implications on more complex collisions. When the contact surface is static but non-planar with spatially varying surface normals, \( \mathbf{J}_n \) is dependent on the positions of the nodes where the contact events occur. In this case, we estimate the contact normal based on the positions where the contact events occur, which are given by any collision detection algorithms. Replacing \( \mathbf{J}_n \) with \( \mathbf{J}_n(\mathbf{x}_n, \mathbf{v}_n) \) requires very minimal extra work to the gradients in Sec. 4.2, as it contributes to the gradients with respect to \( \mathbf{v}_n \) in a straightforward manner. Another type of collisions we consider is self-collisions between nodes. In this case, contact occurs between pairs of nodes where the contact normals are defined by the relative position between two nodes in the pair. Therefore, we can still define \( \mathbf{J}_n \) as a function of \( \mathbf{x}_n \) with each row block now consisting of two blocks of nonzero elements corresponding to the two contact nodes in the pair. Similarly, the gradient derivation remains unchanged except that the dependencies between \( \mathbf{J}_n \) and \( \{\mathbf{x}_n, \mathbf{v}_n\} \) need to be added to Eqn. (23). As we inherit from Ly et al. [2020] the contact model on nodes only, we also inherit its limitation of not handling other types of self-collisions like edge-edge collisions or vertex-face collisions. We leave the derivation of gradients with such cases as future work.

5 EVALUATIONS

This section evaluates and discusses numerical properties of the proposed differentiable cloth simulation method in Sec. 4. We first evaluate its gradients by analyzing their source of non-smoothness and studying their usefulness in high-dimensional optimization problems. Next, we compare the dry frictional model with the contact model in the state-of-the-art differentiable cloth simulation method [Liang et al. 2019] and discuss their differences. Finally, we analyze the numerical properties of our iterative solver in backpropagation and compare its performance with a direct linear solver.

5.1 Continuity and Smoothness

A fundamental assumption in any differentiable simulators is that the underlying physics system is smooth so that gradients can be well-defined. Eqn. (9) contains three possible sources of discontinuities and non-smoothness in the order of their occurrences: determining the contact set \( \mathbf{J}_n \), computing \( \mathbf{J}_n \) that represents the local contact frames, and choosing between three branches from \( C^d_n \) in Eqn. (8). Below we discuss the effects from each of them in detail.

Continuity of branches in contact conditions. First, we empirically demonstrate through an experiment below that switching between the three branches in Eqn. (8) does not create discontinuity in Eqn. (9). In particular, consider the \( n \)-th time step and assume that \( C^d_n \) is fixed and that \( \mathbf{J}_n \) is a continuous function of \( \{\mathbf{x}_n, \mathbf{v}_n\} \), if we treat Eqn. (9) as a function that takes as input \( \{\mathbf{x}_n, \mathbf{v}_n\} \) and returns \( \{\mathbf{x}_{n+1}, \mathbf{v}_{n+1}\} \), we will validate that this function is continuous. In other words, perturbing \( \{\mathbf{x}_n, \mathbf{v}_n\} \) a little will not cause jumps in the resultant \( \{\mathbf{x}_{n+1}, \mathbf{v}_{n+1}\} \) even though the corresponding \( \mathbf{r}_{n+1} \) may need to switch between branches during the perturbation. The intuition is that the three branches in Eqn. (8) together define a contact set \( C^d_n \) in which \( \{\mathbf{r}_{n+1,j}, \mathbf{u}_{n+1,j}\} \) from one branch can smoothly transition to another.

We empirically validate the continuity of branch switching with the following experiment. We simulate a piece of cloth on a rigid, static, and frictional sphere for 200 timesteps. The sphere has one frictional coefficient \( \mu \), and we repeat the experiments by varying \( \mu \) from 0 to 0.35 (Fig. 2). When \( \mu \) is large, all nodes are fixed on the sphere due to their large static friction. When \( \mu \) is close to 0, the cloth slides on the sphere under gravity and takes off from the sphere near the end of the simulation. As \( \mu \) changes from 0.35 to 0, each node on the cloth undergoes the transition from sticking to slipping, and eventually takes off. However, since each node has a different contact normal, the turning point for each of them to switch between these branches is different. Overall, when we gradually change \( \mu \), the ratio among the numbers of nodes with static friction, dynamic friction, and no friction at the end of the simulation also changes gradually, allowing us to observe how switching between these branches affects the continuity of the physical quantities in simulation.

We summarize the quantitative results from this simulation experiment in Fig. 2 (green curves). Specifically, we plot the velocity of three nodes \( A, B, \) and \( C \) as a function of \( \mu \) at an intermediate time step (50) and near the end of simulation (200). We select three nodes
from the corners, edge centers, and the center of the cloth, respectively. All velocities converge to 0 when \( \mu \) becomes large, which is as expected because a large \( \mu \) leads to static friction that freezes these nodes. We also notice a turning point in the velocity curve for each node, indicating a switch between static and dynamic friction. The turning points are located at slightly different \( \mu \) values for each node because their normals on the sphere surface are different. We observe from the figure that the branches in Eqn. (8) do not cause discontinuities.

While the above experiment confirms that these branches do not create discontinuities, we note that branch switching does introduce non-smoothness due to corner cases that can be arbitrarily classified into either branch, e.g., when a node is static but about to slip. Gradients at these corner cases are not well-defined, but the subset these corner cases occupy in \( C_i^n \) has measure zero. Therefore, we still expect gradient-based optimization to be functional, just like we have observed the success of gradient-based optimization in modern deep learning with non-smooth but continuous operators, e.g., ReLU, max pooling, and so on.

**Continuity of local frames at contact nodes.** A second source of possible discontinuity and non-smoothness comes from computing \( J_n \), which consists of two steps: determining a contact point on the contact surfaces and calculating the local normal and tangent vectors. Both steps depend on the geometric representation of the contact surfaces. As pointed out by previous papers in rigid body dynamics [Popović et al. 2000; Werling et al. 2021], contact surface discretization causes jumps in surface normals, and therefore they create discontinuities in velocity and position calculation. To confirm this observation in cloth simulation, we repeat the previous experiment by replacing the analytically described sphere with a triangulated one (pink triangulated sphere surface in Fig. 2 right). After such replacement, we clearly observe jumps in the intermediate and final velocity from the selected contact nodes (pink curves in Fig. 2). Note that the jumps in velocities from the final velocities are less evident than from the intermediate velocities because the vertical axes have different ranges. This observation agrees with similar experiments from previous papers about rigid body dynamics and suggests that one should favor analytical surfaces in differentiable cloth simulation whenever possible.

We end our discussion on the discontinuities from \( J_n \) with two more remarks. First, we notice the contact node velocity curves in Fig. 2 are partitioned into a small number of continuous segments, which is consistent with the result reported by previous work [Popović et al. 2000] discussing contact events on rigid body dynamics. Second, if the scene contains multiple objects the cloth can be in contact with, it is not uncommon to see jumps in the locations of contact events, e.g., from one object to another, even with a continuous representation of each object. Such jumps naturally lead to large discontinuities in simulation. While we do not provide solutions to it in this work, a closely related issue has been extensively studied in differentiable rendering [Li et al. 2018a] from which we may borrow inspirations in the future.

**Continuity of contact sets.** The last and most common source of discontinuity comes from deciding the contact set at each time step, i.e., whether a node should be added to the contact set or not. Obviously, this selection process is not continuous. It is worth noting that having a constant contact set does not cause too much trouble for gradient computation regardless of its size (Fig. 2). Instead, it is the change in the contact set from time to time that brings in discontinuities because whenever a new node is put in the contact set, it adds an impulsive force to the right-hand side of Eqn. (9).

To better understand the effects of changes in contact sets, we hang a piece of cloth above a static sphere in simulation and let the lower half of the cloth fall and slide on the sphere due to gravity (Fig. 3 top). We equip this experiment with a system identification task of the frictional coefficient \( \mu \) and the stiffness parameter \( k \) of the cloth: given a motion sequence of the cloth generated from a pair of unknown \( \mu \) and \( k \), we define a loss function that sums over all time steps the squared distance between each node position in simulation and its corresponding location in the given motion sequence. We repeat the task with four settings of the sphere at different horizontal offsets, leading to a varying frequency of contact events among them.

We plot the landscape of the loss function in Fig. 3 (middle) and compare its smoothness among the four settings with different frequencies of contact events. At first glance, it seems that all four landscapes are equally smooth. However, magnifying a small region of each landscape shows that there is a profound distinction between their continuity and smoothness (Fig. 3 bottom): as establishing and breaking contact becomes more frequent, the local landscape tends to be bumpier.

**Summary.** We have discussed the three sources of potential discontinuities and non-smoothness in our differentiable cloth simulator ordered by their damage to the gradients: the branches in the contact conditions only introduce non-smoothness and maintain continuity; contact surface discretization creates discontinuities due to the jumps of normals across adjacent triangles, but we still expect continuity almost everywhere; adding or deleting nodes in the contact set creates frequent and the most severe discontinuities due to the introduction or removal of impulsive forces.

### 5.2 Usefulness of Gradients

One advantage of gradient-based approaches over their gradient-free counterparts is their faster convergence rate: typically, gradient-free methods explore the local landscape of an objective function by evaluating it with massive samples in the neighborhood. However, such a sampling approach quickly becomes less efficient as the dimension of the decision variables grows higher. Due to this reason, we hypothesize that using gradients from our differentiable simulator will become most beneficial when we have a large number of decision variables. We verify this hypothesis using a control optimization problem: we simulate a piece of cloth with time-invariant external forces applied to each node. The goal is to design the force at each node to pull the center of the cloth to a target position in the end (Fig. 4 top). We simulate the cloth with four settings on its DoFs: 4 × 4 nodes, 5 × 5 nodes, 7 × 7 nodes, and 10 × 10 nodes. Therefore, the number of variables in the optimization problem is 48, 75, 147, and 300, respectively.
We compare the contact model in our differentiable simulator with the one in Liang et al. [2019], which has any contact or self-collisions, whereas every single node of the napkin in “Wind” barely has any contact or self-collisions, whereas every single node of the ribbon in “Slope” is in contact with the plane. We then compare the time cost of our iterative solver with a sparse matrix solver treating $(P - \Delta P - \Delta R)$ as a whole. To understand the performance of the proposed iterative solver, we design two benchmark tests: a “Wind” test where a hanging napkin moves under synthetic wind and a “Slope” test where a ribbon slides on a slanted plane (Fig. 6). These two tests represent two extreme cases in contact handling: the napkin in “Wind” barely has any contact or self-collisions, whereas every single node of the napkin without verifying whether such an update requires sticky contact forces. When the napkin becomes in contact with the concave inner surface of the bowl, such a direct modification injects extra elastic energies. This experiment confirms supporting a more advanced contact and friction model in differentiable cloth simulation is both viable and beneficial.

5.4 Evaluation of Iterative Solver in Backpropagation
Another key component in our differentiable cloth simulation is the iterative solver in Eqn. (29) that utilizes the prefactorized P in PD. In a standard differentiable simulator, solving Eqn. (29) would be done with a sparse matrix solver treating $(P - \Delta P - \Delta R)$ as a whole. To understand the performance of the proposed iterative solver, we design two benchmark tests: a “Wind” test where a hanging napkin moves under synthetic wind and a “Slope” test where a ribbon slides on a slanted plane (Fig. 6). These two tests represent two extreme cases in contact handling: the napkin in “Wind” barely has any contact or self-collisions, whereas every single node of the napkin without numerical explosion. However, the dry frictional contact model in our differentiable simulator shows more physically realistic motion (Fig. 5 middle), whereas the contact model from Liang et al. [2019] leads to more drastic changes in the size of the napkin with a popping artifact after its contact with the bowl (Fig. 5 top and bottom). These artifacts are explainable because the collision handling algorithm in Liang et al. [2019] modifies node positions after penetration without verifying whether such an update requires sticky contact forces. When the napkin becomes in contact with the concave inner surface of the bowl, such a direct modification injects extra elastic energies. This experiment confirms supporting a more advanced contact and friction model in differentiable cloth simulation is both viable and beneficial.

5.3 Benefits of Dry Frictional Contact
We compare the contact model in our differentiable simulator with that in the state-of-the-art differentiable cloth simulator from Liang et al. [2019]. Both models ensure penetration-free simulation and detect self-collisions (node-node collisions in ours and vertex-face and edge-edge collisions in Liang et al. [2019]). However, the contact model in Liang et al. [2019] does not take into account complementarity conditions on either contact forces or frictional forces, which may lead to undesirable artifacts. To visualize this issue, we consider a test scenario in which a napkin falls freely onto the inner surface of a bowl (Fig. 5). The differentiable simulators from both Liang et al. [2019] and our work manage to simulate the napkin without numerical explosion. However, the dry frictional contact model in our differentiable simulator shows more physically realistic motion (Fig. 5 middle), whereas the contact model from Liang et al. [2019]
results. It is also worth mentioning that contact-related operators, threshold and still observe successful gradient-based optimization in Sec. 6, in most of which we use low-precision convergence in the descending direction. This is confirmed empirically by our experiments in Sec. 6, in most of which we use low-precision convergence threshold and still observe successful gradient-based optimization results. It is also worth mentioning that contact-related operators, whose time cost is included in the “Other” and “AR” columns in Table 1, add very little extra overhead to the backpropagation. Finally, we observe uncommon but non-negligible failure cases of the iterative solvers in one experiment (48 × 48 with 1e-6 as the threshold in Table 1) “Wind”. For the 15% failed timesteps, we switch to the sparse LU solver, adding an extra 11.5% time in backpropagation.

6 APPLICATIONS

In this section, we demonstrate a variety of cloth-related applications that can benefit from our proposed differentiable cloth simulation with dry frictional contact. We repeat all experiments with multiple random seeds and use the same random seed set (therefore same initial parameter set) for all methods. We report the optimization results and comparisons with other gradient-free algorithms in Table 2. In Table 3, we report the time steps used by each optimization method to reach minimum final loss as well as the convergence ratio of our iterative solver during back-propagation. For gradient-free algorithms, the number of time steps counts the total steps of forward simulation (each steps the simulation forward in time by \( h \)). For our gradient-based method, we double this number to include the number of back-propagation steps. Note that in practice, back-propagation takes much less time than forward simulation. We have included the wall clock time for running forward simulation and back-propagation of all examples in Appendix A. In Table 2 and all loss vs. time steps plot below, we report the minimum loss or plot

\[
\begin{array}{cccccccc}
A & B & C & D & A & B & C & D \\
\hline
\text{Most Collisions} & & & & \text{Fewest Collisions} & & & \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\mu & k_{\text{stretch}} & \mathcal{L} & \mu & k_{\text{stretch}} & \mathcal{L} & \mu & k_{\text{stretch}} & \mathcal{L} \\
\end{array}
\]

\[
A \quad B \quad C \quad D
\]

\[
\text{Li. et al}
\]

\[
\text{Du et al. 2021; Liu et al. 2017}
\]

\[
\text{PD papers Bouaziz et al. 2014; Du et al. 2021; Liu et al. 2017}
\]

\[
\Delta R
\]

\[
\text{Fig. 3. We simulate collisions between a piece of cloth and a rigid sphere at four different horizontal locations to study the effects of varying numbers of collisions on the smoothness of the simulation. Top: rendering of one of the time steps when the cloth is expected to be in contact with the sphere. The four scenes are ordered by their number of collisions. Bottom: landscapes of the loss defined as a function of the frictional coefficient } \mu \text{ and the stiffness } k \text{ of the cloth (Sec. 5.1). The zoomed-in views of the landscapes show increasing discontinuity and non-smoothness as more collisions occur.}
\]

\[
\text{Fig. 4. Flying Napkin. Comparisons between gradient-based and gradient-free methods for optimizing high-dimensional decision variables in differentiable cloth simulation. Top: the motion sequence of a piece of cloth with external forces parametrized with 300 variables. The goal is to find proper external forces with which the cloth ends at a target position. The loss function is defined as the position discrepancy between the simulated motion sequence and the reference motion. Bottom left: the loss vs. time step curves (300 DoF) from gradient-based (L-BFGS-B) and gradient-free (CMA-ES) methods. Both L-BFGS-B and CMA-ES use the same random seeds. Bottom right: we vary the degrees of freedom in the external force parametrization and repeat the experiment three times. For each experiment, we compute the ratio between the number of time steps used by gradient-free and gradient-based methods until they converge or use up the time budget and plot them as a speedup vs. DoFs curve.}
\]
Table 1. Comparison between the iterative solver (ours) and the sparse LU direct solver in backpropagation under various mesh resolutions in the “Wind” and “Slope” tests. The numbers in the “Res.” column report the mesh resolution. The number in parentheses in the “Solver” column indicates the epsilon value controlling the convergence of the Jacobi solver. The “Backprop Time” column reports the net time in backpropagation and is further decomposed into the next five columns, from which the sum of ratios is 100%. “ΔP” and “ΔR” report the time spent on assembling ΔP and ΔR respectively. “Iter.” and “Direct” shows the time cost by either solver, and operations not covered by these four columns are in the “Other” column. The “Fail” column reports the ratio between the number of timesteps seeing nonconvergence in our iterative solver and the number of total timesteps. The “Speedup” column is the ratio between the direct solver’s time and our time in “Backprop. Time” in each test.

| Test | Res. | Solver | Backprop Time (s) | ΔP (%) | ΔR (%) | Iter. (%) | Direct (%) | Other (%) | Fail (%) | Speedup |
|------|------|--------|------------------|--------|--------|-----------|------------|-----------|----------|---------|
| Wind | 12x12| Direct | 0.606            | 13.9   | -      | 84.4      | 1.7        | 0         | 2.9x     | 1.0x    |
|      |      | Ours (1e-4) | 0.212        | 39.9   | 0      | 55.5      | -          | 4.6       | 0        | 5.7x    |
|      |      | Ours (1e-6) | 0.424        | 19.2   | 78.5   | -         | 2.3        | 0         | 2.3x     | 1.0x    |
|      | 24x24| Direct | 3.360            | 9.9    | -      | 89.1      | 1.0        | 0         | -        | -       |
|      |      | Ours (1e-4) | 0.591        | 58.2   | 0      | 36.3      | -          | 5.5       | 0        | 5.7x    |
|      |      | Ours (1e-6) | 2.858        | 11.7   | 87.2   | -         | 1.1        | 0         | 1.2x     | -       |
|      | 48x48| Direct | 24.001           | 6.8    | -      | 92.5      | 0.7        | 0         | -        | -       |
|      |      | Ours (1e-4) | 2.262        | 63.2   | 0      | 29.7      | -          | 7.1       | 0        | 10.6x   |
|      |      | Ours (1e-6) | 29.255       | 5.4    | 82.6   | -         | 11.5       | 0.6       | 15       | 0.8x    |
| Slope| 12x12| Direct | 0.978            | 13.0   | 3.4    | -         | 82.4       | 1.2       | 0        | 3.1x    |
|      |      | Ours (1e-4) | 0.312        | 70.1   | 9.4    | 16.1      | -          | 4.5       | 0        | 4.5x    |
|      |      | Ours (1e-6) | 0.643        | 18.9   | 2.7    | 76.8      | -          | 1.6       | 0.8x     | 1.5x    |
|      | 24x24| Direct | 5.331            | 10.4   | 1.5    | -         | 87.4       | 0.7       | 0        | -       |
|      |      | Ours (1e-4) | 0.781        | 70.1   | 9.4    | 16.1      | -          | 4.5       | 0        | 6.8x    |
|      |      | Ours (1e-6) | 1.507        | 34.7   | 4.9    | 58.1      | -          | 2.4       | 0        | 3.7x    |
|      | 48x48| Direct | 37.296           | 6.5    | 0.7    | -         | 92.5       | 0.4       | 0        | -       |
|      |      | Ours (1e-4) | 3.095        | 68.1   | 8.2    | 19.8      | -          | 3.8       | 0        | 12.0x   |
|      |      | Ours (1e-6) | 6.825        | 31.2   | 3.5    | 63.6      | -          | 1.8       | 0        | 5.5x    |
Table 2. Comparison between the performance of gradient-based and gradient-free optimization on all examples (except "Hat Controller") in Sec. 6. The "Param #" and "Seed #" columns report the number of optimization parameters and random seeds used in the experiments, respectively. All methods start the optimization with the same initial random seed set. The "Min Init. Loss" column reports the minimum initial loss across all random seeds. The "Optimized Loss Percentage" column reports the optimized loss as a percentage (0-100%) of the minimum initial loss reached across all random seeds for each method: "Final" reports the loss reached when the optimization stops, and "Equal Step #" reports the loss reached by CMA-ES and (1+1)-ES using the same number of time steps for L-BFGS-B convergence. We color the values of loss percentage using a green-orange color scale: green corresponds to 0% and orange corresponds to 100%.

| Name  | Param # | Seed # | Min Init. Loss | L-BFGS-B (Ours) | CMA-ES | (1+1)-ES |
|-------|---------|--------|---------------|----------------|--------|----------|
|       |         |        |               | Final (%)      | Final (%) | Equal Step # (%) | Final (%) | Equal Step # (%) |
| T-shirt | 6       | 5      | 6.62          | 0.53           | 0.60   | 15.78    | 1.74       | 2.43          |
| Sphere | 1       | 10     | 0.46          | 0.43           | 0.00   | 0.43     | 0.65       | 0.65          |
| Hat    | 18      | 5      | 21.83         | 0.42           | 0.55   | 37.51    | 0.51       | 5.87          |
| Sock   | 36      | 5      | 17.08         | 11.59          | 17.39  | 64.70    | 18.43      | 52.80         |
| Dress  | 2       | 16     | 0.90          | 76.91          | 79.02  | 80.02    | 77.25      | 82.57         |
| Flag   | 8       | 10     | 2.88          | 4.77           | 10.57  | 38.89    | 25.57      | 40.14         |

Fig. 5. Bowl. We simulate the motion sequence of the napkin (50 × 50 × 2 triangles, \( h = 5 \text{ms}, 500 \text{time steps} \)) with the contact models from two differentiable cloth simulators: Liang et al. [2019] (top) and ours (middle). The area ratio (bottom) is the ratio between the napkin’s area at the current timestep and in the rest shape. Refer to our video for the full motions.

visually identical to the given input. However, this optimization problem is special in that it only has one parameter and the landscape of the loss function is bumpy due to the large variation in the collision set as a function of the frictional coefficient, as suggested by the Continuity of contact sets experiment in Sec. 5.1. In this case, evolutionary strategies can achieve a lower final loss because the samples can freely explore the full range of frictional contact, while L-BFGS-B can get trapped in a local optimum.

6.2 Robot-Assisted Dressing

Another line of research that can benefit from a differentiable cloth simulator is robot-assisted dressing. The mainstream solution to these tasks is typically gradient-free methods like evolutionary strategies, reinforcement learning, or inverse dynamics before a differentiable cloth simulator becomes available. We present two examples to demonstrate the usage of gradients in robot-assisted dressing: "Hat" (Fig. 9) and "Sock" (Fig. 10). In both examples, the goal is to find trajectories for a kinematic robotic manipulator to put on the hat or the sock. The end effectors of the manipulator pick a few prespecified vertices on the cloth meshes and pull them along the kinematic trajectories, which are parametrized as B-splines. By optimizing the parameters of the B-splines (18 DoFs in "Hat" and 36 DoFs in "Sock"), we can direct the manipulator to move the hat until it reaches the target location on top of the sphere. We define the loss function in "Hat" as the L2-distance between the hat’s final position at the last time step and a predefined target position, which we generate by translating the hat’s rest shape onto the top of the sphere. The loss function for "Sock" is defined as the L2-distance...
Table 3. The “Convergence Time Steps” column reports the number of simulation time steps used for each optimization method until convergence on all examples shown in Sec. 6. For a fair comparison, the time steps shown for L-BFGS-B include both forward simulation and backward propagation time. The “Iter. Conv. %” column reports the percentage of time steps in backward propagation where the iterative solver converges.

| Name   | L-BFGS-B | CMA-ES | (1+1)-ES | Iter. Conv. % |
|--------|----------|--------|----------|---------------|
| T-shirt| 4500     | 53500  | 47750    | 99.75         |
| Sphere | 2450     | 18900  | 11200    | 100.00        |
| Hat    | 11200    | 159200 | 106000   | 90.58         |
| Sock   | 17600    | 176800 | 91600    | 99.41         |
| Dress  | 2750     | 3125   | 10000    | 84.80         |
| Flag   | 6800     | 17300  | 24800    | 96.00         |

Fig. 7. **T-shirt** (4278 DoFs, $h = 1/90s$, 250 time steps). Estimating the cloth material and wind parameters based on a given synthetic motion sequence of the T-shirt. From left to right we show the simulated T-shirt at 0, 80, 160, and 240 time steps. Top: the ground-truth motion sequence; Middle: simulation with the initial guess on the cloth and wind parameters; Bottom: simulation after optimization with L-BFGS-B.

between the desired and simulated locations of a few key points manually chosen on the sock and evaluated at the middle and the end of the simulation.

With the gradient information at hand, we run the L-BFGS-B optimizer to tune the parameters of the trajectories and compare its performance with CMA-ES and (1+1)-ES (Table 2 and 3). We notice that within the same time steps, L-BFGS-B converges substantially faster than the gradient-free baselines to a better solution (Fig. 9 and Fig. 10). We can safely conclude that the fast convergence of L-BFGS-B unlocked by our differentiable cloth simulator is a clear advantage over gradient-free methods.

Fig. 8. **Sphere** (1875 DoFs, $h = 1/180s$, 350 time steps). Estimating the frictional coefficient $\mu$ between the sphere and the cloth based on the cloth’s contact with the sphere. From top to bottom we show the simulation at 0, 100, 200, and 300 time steps. Top: the ground-truth motion sequence; Middle: simulation with the initial guess on $\mu$; Bottom: simulation after optimization with L-BFGS-B.

Fig. 9. **Hat** (1737 DoFs, $h = 1/100s$, 400 time steps). Optimizing trajectories for a manipulator to move a hat onto the sphere. Top left: Initial trajectory from one random seed before optimization overlaid with intermediate hat positions in simulation. Top right: the optimized trajectory from L-BFGS-B (which shares a visually similar trajectory with the ones optimized by ES algorithms). Bottom: The loss vs. time step curves for all methods.
6.3 Inverse Design
The next application we present is "Dress", an inverse design example that aims to optimize cloth material parameters in a dress so that its dynamic motion can satisfy certain design intents. Specifically, we optimize the material parameters of a twirl dress so that after the dress spins, the apex angle of the cone-like dress agrees with the target value (100 degrees in our experiment). We define the loss function as the difference between the hemline height corresponding to the target apex angle and the estimated apex angle from points on the hemline of the dress at the last frame of the simulation. We report the optimization results from L-BFGS-B and two ES baselines in Table 2 and visualize the simulation results before and after optimization in Fig. 1. Similar to the previous tasks, we notice that L-BFGS-B achieves better optimized results using fewer time steps.

6.4 A Real-to-Sim Example
In this section, we present a real-to-sim "Flag" example (Fig. 11). In this example, we use the real-world motion sequence captured on a flag flapping in the wind from previous work [White et al. 2007] and aim to reconstruct a digital twin of the scene in simulation. This includes not only estimating the material parameters of the flag but also modeling the unknown wind condition at the capture time, which is particularly challenging due to its intricate stochastic model with unknown degrees of freedom. We model the wind force at each time step as a 3D force applied near the center of the scene and spatially decaying proportional to the inverse distance to the center. To model the transient nature of the wind force, we modulate magnitude of the 3D vector of a sinusoidal wave as a function of time with parameterized frequency and phase offset. Together, the material and wind model define an eight-dimensional parameter space to be optimized. We define the loss function as the L2-distance between the positions of all nodes at each time step in simulation and the ground-truth motion sequence.

We solve this task using L-BFGS-B and the two ES methods and report their performance in Table 2, 3, and Fig. 11. All three methods substantially reduce the loss after optimization, but L-BFGS-B achieves a lower final loss. We plot the trajectories of 6 key nodes before and after optimization (orange) along with the ground-truth reference trajectories (yellow) from the motion-captured data in Fig. 11. By comparing the left and right images in Fig. 11, we can see that the L-BFGS-B optimizer reduces the discrepancy substantially between the simulated and actual trajectories after optimization. The real-to-sim matching is still imperfect as indicated by the nonzero final loss, which we suspect could be due to the simplistic nature of our synthetic wind model. A more sophisticated and expressive wind model, e.g., a neural network, may serve a better role in modeling and matching the real-world physics, which we leave as future work.

Fig. 10. Sock (3165 DoFs, \( h = 1/160 s, 400 \) time steps). Optimizing trajectories for a manipulator to put a sock onto the foot model. Top left: One initial trajectory before optimization. We show the intermediate time steps on the left and the final state of the sock on the right. Top right: one control trajectory optimized by L-BFGS-B. The end effectors successfully put the sock onto the foot using the optimized trajectory. Bottom: the loss vs. time step curves for all methods.

Fig. 11. Flag. A real-to-sim example that reconstructs a digital flapping flag (540 vertices, 1026 triangles, \( h = 1/120 s, 100 \) time steps) based on motion data captured from real-world experiments. Top: We plot the trajectories of 6 nodes on the cloth from the ground-truth motion (yellow curves) and the simulation results with guesses on the material and wind parameters before optimization (orange curves, left) and after optimization (orange curves, right). Bottom: The loss vs. time step curves for all methods.

6.5 Hat Controller
We end this section with an advanced "Hat" task. Unlike the previous open-loop trajectory optimization with a fixed starting position, the goal of this new task is to train a generalizable closed-loop controller that can put on the hat from a random starting position sampled from a fixed-radius hemisphere around the head. Specifically, we train a closed-loop control policy \( a_t = \pi_\theta(s_t) \), which takes as input the current state \( s_t \) of the task and outputs an action vector \( a_t \) at each time step \( t \). The state vector \( s_t \) includes the hat node positions,
We train a closed-loop controller for the advanced “Hat” task (Sec. 6.5) which aims to move the hat from different initial positions (sampled from a fixed-radius hemisphere) onto the head. Top two rows: We visualize the control trajectories of the hat in ten initial positions denoted by their elevation and azimuth angle at the bottom of each subfigure. Bottom: The loss vs. time step curve for our gradient-based optimization and PPO.

In each epoch during training, we randomly sample 20 starting positions of the hat and compute a loss averaged from all simulation sequences. The loss function of each sequence is defined by $L = L_{\text{deform}} + L_{\text{target}} + L_{\text{dir}}$, where $L_{\text{deform}}$ measures the stretching of the hat using the distance change between the two end-effectors, $L_{\text{target}}$ measures the L2-distance between the last-time-step pose of the hat and the target pose, and $L_{\text{dir}}$ is the orientation difference between the last-time-step pose and the target pose of the hat. The gradient of the loss is then computed by our differentiable simulation framework and used by a gradient-based optimizer (Adam [Kingma and Ba 2017]) to update the policy parameters. For testing, we evaluate the controller from 20 fixed starting position configurations uniformly sampled on the hemisphere surface.

To compare our method with gradient-free methods, we also solve the task with Reinforcement Learning (RL), which has been widely used to train complex neural network policies for robot-assisted dressing problems [Clegg et al. 2018]. Specifically, we compare our gradient-based method with PPO [Schulman et al. 2017], a state-of-the-art RL algorithm. We similarly sample a starting position from the hemisphere in each iteration when training PPO. For a fair comparison, we design the reward function $r_t$ to be the sum of the negative counterpart of $L$ plus a constant to avoid negative rewards, and observation of the environment to be $s_t$. We evaluate Adam and PPO using $L$ as the common metric and plot the optimization curve at the bottom of Fig. 12. We see that both methods reach a similar final loss, but with our differentiable simulation framework, the gradient-based method reaches its final loss with a 85x speedup (Adam uses 23200 time steps; PPO uses 1978000 time steps).

After the training process converges, both our gradient-based method and PPO successfully move the hat onto the head from all 20 testing positions. We visualize the trajectories generated by
Adam’s trained policy from 10 testing starting positions at the top of Fig. 12. Unlike existing differentiable simulation papers [Du et al. 2021; Hu et al. 2020, 2019; Liang et al. 2019] that train a closed-loop network controller only for a fixed state, we highlight that we use differentiable simulation to train the network from multiple, random states and study its generalizability in a test set of unseen states. This allows us to conduct a fairer comparison between gradient-based optimization method and RL methods, which is typically overlooked in previous papers.

7 CONCLUSIONS, LIMITATION, AND FUTURE WORK
In this paper, we presented a differentiable cloth simulator built on PD with Signorini-Coulomb frictional contact. Our differentiable simulator is different from existing papers in its simultaneous accommodation of rich and frequent (self-)contact, Signorini-Coulomb contact law, and differentiability in cloth simulation. We analyzed the numerical properties of gradients from our differentiable simulator, including the source of discontinuities and its empirical speedup over gradient-free approaches in high-dimensional problems. We additionally presented an iterative solver that exploits the contact gradients to speed up the backpropagation and observed a substantial speedup (up to an order of magnitude with low-precision simulation) over direct solvers. Our differentiable cloth simulator enabled gradient-based optimization methods in a diverse set of applications, for which traditional gradient-free methods are generally much less sampling efficient. In particular, we presented a preliminary study on training a generalizable closed-loop controller using differentiable simulation, in which our approach and PPO achieved comparable performance and generalizability, but we used much less time.

There are still quite a few limitations in our method that are worth further investigation. First, since our method is built on PD, it also limits the choice of material models. It would be useful to generalize the current framework and support more physically accurate cloth material models, e.g., the piecewise linear elastic model described in Wang et al. [2011].

Second, the contact model we use from Ly et al. [2020] does not take into account vertex-face or edge-edge self-collisions. While we empirically observed that handling only vertex-vertex self-collisions managed to produce plausible results with medium mesh resolution, vertex-face and edge-edge collision detection and handling is still highly desirable for a more physically realistic cloth simulator.

Third, although we have identified some possible sources of discontinuities and non-smoothness in our differentiable simulator with empirical experiments, their effects on gradient-based optimization still require a thorough investigation. In particular, the locally bumpy energy landscape we observed in Fig. 3 due to changes in contact sets makes us question both the necessity and the usefulness of exact gradients in optimization, although Sec. 6 implies that gradients were still helpful in many downstream applications. Noting that the bumpiness in Fig. 3 is local and the global view of the energy landscape is still smooth, we hypothesize an inexact but smoothed gradients would be more powerful, which we leave as future work.

Fourth, there is no theoretical guarantee on the convergence of the iterative solver we implemented in backpropagation. Although non-convergence is uncommon in our experiments, it introduces a costly switch to the slower direct solver, which we hope to fully resolve in future work.

Finally, many of our applications were in simulation only. It would be more exciting if these results could be replicated in real-world settings. We consider connecting our differentiable cloth simulator to more real-world applications, including real-world robot-assisted dressing, material parameter identification for real-world fabric samples, computational design for sports suits, and so on. Closing the sim-to-real gap is a nontrivial problem, in which we believe our differentiable simulator could play a beneficial role.

ACKNOWLEDGMENTS
We thank Marco Renedo for his helpful discussions on the preconditions. Junbang Liang for his help with running the baseline comparison code, and the anonymous reviewers for their helpful comments. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grant No. FA8750-20-C-0075.

REFERENCES

David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH ’98). Association for Computing Machinery, New York, NY, USA, 43–54.

Aric Bartle, Alla Sheffer, Vladimir G. Kim, Danny M. Kaufman, Nicholas Vining, and Flo-raine Berthouzoz. 2016. Physics-Driven Pattern Adjustment for Direct 3D Garment Editing. ACM Trans. Graph. 35, 4, Article 50 (July 2016), 11 pages.

Sofien Bouaziz, Sebastian Martin, Tianjian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. ACM Trans. Graph. 33, 4, Article 154 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601116

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. ACM Trans. Graph. 21, 3 (July 2002), 594–603.

R. Bridson, S. Marino, and R. Fedkiw. 2003. Simulation of Clothing with Folds and Wrinkles. In Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (San Diego, California) (SCA ’03). Eurographics Association, Goslar, DEU, 28–36.

Bernard Brogliato. 2016. Nonsmooth Lagrangian Systems. Springer International Publishing, Cham, 241–370.

Remi Brouet, Alla Sheffer, Laurence Boissieux, and Marie-Paule Cani. 2012. Design Preserving Garment Transfer. ACM Trans. Graph. 31, 4, Article 6 (July 2012), 11 pages.

George E. Brown, Matthew Overby, Zahra Forootaninia, and Rahul Narain. 2018. Accurate Dissipative Forces in Optimization Integrators. ACM Trans. Graph. 37, 6, Article 282 (Dec. 2018), 14 pages.

Kwang-Jin Choi and Hyeong-Seok Ko. 2002. Stable but Responsive Cloth. ACM Trans. Graph. 21, 3 (July 2002), 608–611.

Alexander Clegg, Zackory Erickson, Patrick Grady, Greg Turk, Charles C. Kemp, and C. Karen Liu. 2020. Learning to Collaborate From Simulation for Robot-Assisted Dressing. IEEE Robotics and Automation Letters 5, 2 (2020), 2746–2753.

Alexander Clegg, Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. 2018. Learning to Dress: Synthesizing Human Dressing Motion via Deep Reinforcement Learning. ACM Trans. Graph. 37, 6, Article 179 (Dec. 2018), 10 pages.

David Clyde, Joseph Teran, and Rasmus Tamstorf. 2017. Modeling and Data-Driven Parameter Estimation for Woven Fabrics. In Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (Los Angeles, California) (SCA ’17). Association for Computing Machinery, New York, NY, USA, Article 17, 11 pages.

C. Dario Bellalcoco, Christian Gehring, Jemin Hwangbo, Péter Farkásháusser, and Marco Hutter. 2016. Perception-less terrain adaptation through whole body control and hierarchical optimization. In 2016 IEEE-RAS 16th International Conference on humanoid Robots (Humanoids). 558–564. https://doi.org/10.1109/HUMANOIDS.2016.780330

Fulge de Avila, Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. 2018a. End-to-end differentiable physics for learning and control. Advances in neural information processing systems 31 (2018), 7178–7189.
We run all optimizations on a workstation of 80 CPU cores and 80G memory. Depending on the problem complexity, the wallclock time of running these optimizations varies from less than 30 minutes to 2 hours for all methods. We report the run time for optimizing the examples shown in Sec. 6 using our gradient-based method in Table 4.

### B.1 System Identification

#### T-shirt.

The loss function is defined as

\[
L = \sum_{n=1}^{N} \| x_n^{\text{current}} - x_n^{\text{target}} \|_2^2, \tag{31}
\]

where \( N = 240 \) is the total number of time steps, \( x_n^{\text{current}} \) is the position of the mesh during optimization, and \( x_n^{\text{target}} \) is the position of the ground-truth simulation generated using our system. There are 4 parameters (6 DoFs) to be optimized: \( \theta_{\text{T-shirt}} = ([k_{\text{stretch}}; \theta; \omega; d]) \), where \( k_{\text{stretch}} \) controls the stretching stiffness of the cloth material and \( \theta, \omega, d \) controls the parameterized external wind force: each node receives a three-dimensional wind force \( 0.5 |\sin(\omega t + \phi)| + 1.0 |d| \).

#### Sphere.

The loss function is defined as the same as in the “T-shirt” example above except that the total time \( N = 300 \). The decision variable is the 1-DoF frictional coefficient of the sphere.

### B.2 Robot-Assisted Dressing

#### Hat.

The loss function is defined as the L2 norm between the last-time-step position of the hat \( x_N^{\text{current}} \) and a target position \( x_N^{\text{target}} \) generated by translating the initial pose of the hat onto the head:

\[
L = \| x_N^{\text{current}} - x_N^{\text{target}} \|_2^2, \tag{32}
\]

where \( N = 400 \) is the total number of time steps. The trajectory of each of the two end effectors is controlled by a cubic Hermite spline.

Each spline has 3 parameters (9 DoFs): \( \theta_{\text{spline}} = (t_1, t_2, p_{\text{end}}) \), where \( t_1 \) and \( t_2 \) are the two tangents of the spline and \( p_{\text{end}} \) is the endpoint of the spline.

#### Sock.

The goal is to optimize for the trajectory of the three end effectors holding onto a sock so that the sock can be put onto a synthetic foot model from an initial starting position. To guide the end effectors to first hook the opening of the sock onto the tip of the foot, then slide the sock upward onto the leg, the loss function is designed as

\[
L = \sum_{n \in \{2, \ldots, N\}} \sum_{(p_{\text{foot}}, p_{\text{sock}}) \in \mathcal{P}_n} \| x_{n, \text{foot}}^{\text{target}} - x_n^{\text{current}} \|_2^2, \tag{33}
\]
where \( \mathcal{P}_t \) defines a set of manually selected keypoint correspondence pairs \((p_{\text{foot}}, p_{\text{sock}})\) between the sock mesh \( x^n \) and the foot model \( x^N \) at halfway \( t = \frac{N}{2} \) and the end of the simulation \( t = N \), respectively. We sum up the L-2 norm of the position difference between each correspondence. The optimization parameters are the Hermite spline parameters for each of the four end effectors defined similarly as in the "Hat" example (36 DoFs in total).

### 3 Inverse Design

#### Dress

The loss function is defined as

\[
L = \sum_{p \in \mathcal{P}} (h_p - h)^2, \tag{34}
\]

where \( h \) is the calculated target height of the bottom of the dress when the desired apex angle is reached. \( \mathcal{P} \) is the set of all points located at the bottom of the dress, and \( h_p \) is the height of the point \( p \) (corresponding to the y coordinate of the point in our implementation). There are 2 parameters (2 DoFs) that are being optimized: \( \theta_{\text{Dress}} = (d, k_{\text{bend}}) \) where \( d \) is the density of the fabric and \( k_{\text{bend}} \) is the bending stiffness of the fabric.

#### Real-to-Sim Example

In this task, we optimize for the material parameters of the flag and the parameters of a simple wind model to match the motion trajectory of a flag to real-world captured data. The wind model is defined as

\[
f_{\text{ext}} = \frac{\sin(\omega t + \phi) + 1}{2} k d, \tag{35}\]

where \( d \) is a 3D vector to be optimized and \( k \in \mathbb{R}^m \) a constant coefficient vector with each entry equal to a node’s inverse distance to the flag center evaluated at the first time step. There are 8 parameters to be optimized \( \theta_{\text{flag}} = (k_{\text{stretch}}, k_{\text{bend}}, \rho, \phi, \omega, d) \), where \( k_{\text{stretch}} \) and \( k_{\text{bend}} \) are the stretching and bending stiffness of the fabric, \( \rho \) the density of the fabric, and \( \phi, \omega, d \) the parameters of the wind model defined above.

#### Hat Controller

In this task, the goal is to optimize for the neural network parameters of the hat controller so that the two end effectors put a hat on a head model from any starting position defined on a fixed-radius hemisphere centered at the head model. During training, we uniformly sample 20 starting positions on the hemisphere for each epoch, and compute the loss averaged from all simulation sequences. We define the loss function as

\[
L = L_{\text{deform}} + L_{\text{target}} + L_{\text{dir}}. \tag{36}
\]

\( L_{\text{deform}} \) minimizes the distance change between the two end effectors and is defined as \( L_{\text{deform}} = \sum_{n=1}^{N} \|x_{n} - x_{n,e1}\|_2 \) where \( e_1 \) and \( e_2 \) are the indices of the nodes pulled by the two end effectors. \( L_{\text{target}} \) minimizes the L2-distance between the poses of the hat and the target pose at the last few frames (20 in our implementation) and is defined as

\[
L_{\text{target}} = \sum_{n=N-20}^{N} \|x_{n} - x_{n}^{\text{target}}\|_2. \tag{37}
\]

\( L_{\text{dir}} \) minimizes the orientation difference between the last-time-step pose and the target pose of the hat and is defined as

\[
L_{\text{dir}} = d_{\text{target}} \cdot d_{\text{current}}. \tag{38}\]

where \( d_{\text{target}} \) is the up vector of the hat at the target pose and \( d_{\text{current}} \) is defined similarly to the hat at the last time step.

### C Optimization Results for All Random Seeds

In this section we report the optimization results for all random seeds in Table 5 and Table 6. For most experiments, L-BFGS-B achieves lower or comparable optimized loss than the gradient-free methods using a factor (often to an order of magnitude) of time steps, thanks to the gradient information provided by our differentiable simulator. For some random seeds, L-BFGS-B converges to a relatively large final loss percentage, possibly due to being stuck in a local minimum. In practice, it is common and recommended to run gradient-based optimizations with several initial seeds to alleviate this problem, and is the rationale behind why we choose to report the minimum loss achieved across all random seeds in the results shown in Table 2 and plot the minimum loss envelop in Fig 4, 9, 10, 11. It is also worth mentioning that the examples shown Sec. 6 have a relative low number of design variables, while the speed-up of gradient-based methods becomes even more evident with more design variables as demonstrated by the "Flying Napkin" experiment in Fig. 4.
Table 5. We report the optimization results for all random seeds of all methods in the “T-shirt” and “Flag” examples. For each random seed, we report its initial loss and final loss achieved by each optimization method. “Final Loss Percentage (%)” reports the optimized loss as a percentage (0-100%) of the initial loss. “Convergence Time Steps” reports the number of time steps used by each method to reach its final loss. For all tasks, we also summarize the minimum, average and median statistics across all random seeds for each column. For each metric (“Final Loss”, “Final Loss Percentage (%)”, “Convergence Time Steps”) and each row, the minimum number across the three optimization methods is marked in bold.

| Initial Loss | Final Loss | Final Loss Percentage (%) | Convergence Time Steps |
|--------------|------------|---------------------------|------------------------|
|              | L-BFGS-B   | CMA-ES | (1+1)-ES | L-BFGS-B | CMA-ES | (1+1)-ES | L-BFGS-B | CMA-ES | (1+1)-ES |
| T-shirt      |            |        |            |          |        |            |          |        |            |
| MIN          | 0.035      | 0.052  | 0.115      | 0.1      | 0.2    | 0.6        | 2500     | 41750  | 42000      |
| AVERAGE      | 1.350      | 0.171  | 0.430      | 20.0     | 1.0    | 2.1        | 4300     | 34800  | 31200      |
| MEDIAN       | 0.042      | 0.169  | 0.213      | 0.2      | 0.8    | 1.6        | 3250     | 27000  | 19750      |
| Flag         |            |        |            |          |        |            |          |        |            |
| MIN          | 0.137      | 1.118  | 1.154      | 4.8      | 38.9   | 40.1       | 3400     | 17300  | 24800      |
| AVERAGE      | 1.136      | 0.945  | 1.079      | 29.6     | 24.6   | 28.1       | 1100     | 900    | 27600      |
| MEDIAN       | 1.175      | 0.304  | 1.021      | 28.9     | 7.5    | 25.1       | 5200     | 1200   | 28500      |

ACM Trans. Graph., Vol. 1, No. 1, Article 1. Publication date: January 2022.
| Initial Loss | Final Loss | Final Loss Percentage (%) | Convergence Time Steps |
|--------------|------------|----------------------------|------------------------|
|              | L-BFGS-B   | CMA-ES                     | (1+1)-ES               | L-BFGS-B   | CMA-ES                     | (1+1)-ES |
| **Hat**      |            |                            |                        |            |                            |          |
| 54.60        | 10.391     | 0.754                      | **0.330**              | 19.0       | 1.4                        | **0.6**  |
| 33.81        | 0.402      | 0.665                      | **0.105**              | 1.2        | 1.9                        | **0.3**  |
| 43.45        | 0.091      | 0.723                      | 0.236                  | **0.2**    | 1.6                        | 0.5      |
| 48.63        | 0.105      | 0.283                      | 1.290                  | **0.2**    | 0.6                        | 2.6      |
| 21.83        | 0.096      | 0.946                      | 0.314                  | **0.4**    | 4.3                        | 1.4      |
| **MIN**      | **0.091**  | 0.283                      | 0.105                  | **0.2**    | 0.6                        | 0.3      |
| **AVERAGE**  | 2.217      | 0.674                      | **0.455**              | 4.2        | 2.0                        | **1.1**  |
| **MEDIAN**   | **0.105**  | 0.723                      | 0.314                  | **0.4**    | 1.6                        | 0.6      |
| 46.02        | 1.980      | 7.267                      | 5.335                  | **4.3**    | 15.8                       | 11.6     |
| 41.22        | 3.243      | 6.131                      | 8.396                  | **7.9**    | 14.9                       | 20.4     |
| 39.10        | 8.856      | 10.547                     | **3.047**              | 22.6       | 27.0                       | 7.8      |
| 17.08        | 2.589      | 4.149                      | 7.830                  | **15.2**   | 24.3                       | 45.8     |
| 33.29        | 2.652      | 4.126                      | 5.791                  | **8.0**    | 12.4                       | 17.4     |
| **MIN**      | **1.980**  | 4.126                      | 3.047                  | **4.3**    | 12.4                       | 7.8      |
| **AVERAGE**  | 3.864      | 6.444                      | 6.080                  | **11.6**   | 18.9                       | 20.6     |
| **MEDIAN**   | **2.652**  | 6.131                      | 5.791                  | **8.0**    | 15.8                       | 17.4     |
| **Sock**     |            |                            |                        |            |                            |          |
| 0.46         | 0.002      | **0.000**                  | 0.003                  | 0.4        | 0.0                        | 0.6      |
| 0.90         | 0.064      | **0.000**                  | **0.000**              | 7.2        | 0.0                        | 0.0      |
| 0.58         | 0.002      | **0.000**                  | **0.000**              | 0.3        | 0.0                        | 0.0      |
| 2.20         | 0.904      | **0.000**                  | **0.000**              | 41.1       | 0.0                        | 0.0      |
| 4.03         | 0.904      | 0.004                      | 0.031                  | 22.4       | 0.0                        | 0.8      |
| 0.90         | 0.903      | **0.000**                  | **0.000**              | 100.0      | 0.0                        | 2.0      |
| 0.90         | 0.002      | 0.002                      | **0.000**              | 0.2        | 0.2                        | **0.0**  |
| 0.89         | 0.893      | **0.000**                  | **0.000**              | 100.0      | 0.0                        | 0.8      |
| 0.88         | 0.861      | **0.000**                  | **0.000**              | 97.6       | 0.0                        | 0.1      |
| 0.85         | 0.514      | **0.000**                  | **0.000**              | 60.4       | 0.0                        | 0.0      |
| **MIN**      | **0.002**  | **0.000**                  | **0.000**              | 0.2        | 0.0                        | 0.0      |
| **AVERAGE**  | 0.505      | **0.000**                  | 0.006                  | 43.0       | 0.0                        | 0.4      |
| **MEDIAN**   | 0.688      | **0.000**                  | 0.001                  | 31.7       | 0.0                        | 0.1      |
| **Sphere**   |            |                            |                        |            |                            |          |
| 2.41         | 1.820      | **0.712**                  | 0.845                  | 75.4       | 33.3                       | 39.6     |
| 1.35         | 0.716      | 0.830                      | **0.696**              | 53.2       | 60.4                       | **50.6** |
| 1.57         | 1.406      | 0.824                      | **0.782**              | 89.3       | 52.3                       | 49.6     |
| 1.03         | 0.841      | 0.825                      | **0.822**              | 82.0       | 80.3                       | **80.0** |
| 0.90         | 0.880      | 0.824                      | **0.823**              | 97.7       | 90.3                       | 90.1     |
| 1.49         | 1.490      | 0.832                      | **0.698**              | 99.9       | 55.4                       | 46.5     |
| 1.09         | 0.875      | **0.828**                  | 0.830                  | 80.3       | 75.2                       | 75.4     |
| **MIN**      | **0.693**  | 0.861                      | 0.792                  | **37.0**   | 45.4                       | 41.8     |
| **AVERAGE**  | 1.107      | **0.818**                  | **0.808**              | 78.4       | 61.7                       | **60.9** |
| **MEDIAN**   | 1.029      | **0.824**                  | **0.823**              | 81.1       | 61.3                       | 61.7     |
| **Dress**    |            |                            |                        |            |                            |          |
| **MIN**      | **0.693**  | 0.712                      | 0.696                  | 37.0       | 33.3                       | 39.6     |
| **AVERAGE**  | 1.107      | **0.818**                  | **0.808**              | 78.4       | 61.7                       | **60.9** |
| **MEDIAN**   | 1.029      | **0.824**                  | **0.823**              | 81.1       | 61.3                       | 61.7     |

Table 6. Similar table as Table 5 for the "Hat", "Sock", "Sphere" and "Dress" examples.