# DIFFERENTIATE THE SOLVER, NOT THE EQUATION: REVERSE-SWEEP ADJOINTS FOR BLOCK IMPLICIT SIMULATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Differentiable simulation is a key component in learning, control, and inverse problems, where gradients through nonlinear implicit solvers are required. Existing approaches either rely on unrolled automatic differentiation, whose memory grows with solver depth, or on equation-level implicit differentiation, which assembles global Jacobians and solves large sparse adjoint systems, discarding the locality of the forward solver, and differentiating the *converged* equation rather than the finite computation that actually ran. To overcome these limitations, we propose *solver-level differentiation*, which differentiates the executed solver itself. Our key insight is that when a solver is composed of block implicit updates, its discrete adjoint is obtained by applying the corresponding adjoint updates in reverse order, yielding a reverse-sweep formulation whose backward pass mirrors the forward solver. From an operator perspective, the forward pass realizes an approximate inverse through ordered local solves, and the backward applies its transpose through reverse local adjoint solves, constructing no global system. We instantiate this idea on Vertex Block Descent (Chen et al., 2024a), yielding a differentiable solver whose reverse colored Gauss–Seidel sweeps are composed entirely of local $3\times3$ adjoint solves. The backward matches automatic differentiation through the identical executed forward to machine precision at every solver depth, where the equation-level adjoint is off by $37\%$ after one sweep; in a controlled same-codebase, same-GPU comparison it is $33\times$ faster and uses $69\times$ less memory than unrolled automatic differentiation; and the same construction is exact on projective dynamics and extended position-based dynamics. We scale differentiable elastodynamics to $10^6$ contact-coupled soft bodies (8M vertices) on one GPU. Overall, this work highlights solver structure as a practical organizing principle for efficient differentiable simulation.

## 1 INTRODUCTION

Implicit integrators are the workhorse of stiff physical simulation (Baraff & Witkin, 1998). Each step solves a nonlinear stationary problem

$$\nabla G(\boldsymbol{x}) = \mathbf{0}, \qquad G(\boldsymbol{x}) = \tfrac{1}{2h^2}(\boldsymbol{x}-\boldsymbol{y})^\top M(\boldsymbol{x}-\boldsymbol{y}) + E(\boldsymbol{x}), \tag{1}$$

where $G$ is the incremental potential of the step, $\boldsymbol{x} \in \mathbb{R}^{3|V|}$ stacks the positions of $|V|$ mesh vertices, $M$ is the mass matrix, $h$ the time step, $\boldsymbol{y}$ the inertial prediction from the previous step, and $E$ the elastic potential. Differentiating through such a step is what makes simulation usable as a layer: for system identification, control and trajectory optimization, and training with a simulator in the loop (Hu et al., 2020; 2019; Degrave et al., 2019).

The obstacle is memory. **Unrolled automatic differentiation (AD)** (Griewank & Walther, 2008) traces every elementary operation of the executed program, so the tape grows with solver depth $\times$ rollout length $\times$ mesh size; backpropagation through time (Werbos, 1990) over long rollouts then needs checkpointing. It is exact (we use it as ground truth) but the first to run out of memory. **Equation-level implicit differentiation** (Giles & Pierce, 2000; Blondel et al., 2022) instead discards the executed solver, asserts $\nabla G(\boldsymbol{x}^\star) = \mathbf{0}$, and applies the implicit function theorem, requiring assembly and solution of a global adjoint system. Memory becomes $O(1)$ in solver depth, but a



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

global linear-algebra object reappears; less often noted, the gradient it returns is the derivative of a *converged* solve, not of the computation that ran when the solver executes a small fixed number of sweeps, as it does in practice for speed. The question is *which function the backward should differentiate: the equation the solver approximates, or the finite solver that ran?*

We take the executed solver seriously as the object to differentiate. Many modern simulators build their implicit steps from structured local updates; we build on the *block-coordinate* instance, Vertex Block Descent (Chen et al., 2024a), which minimizes Eq. (1) by sweeping over small blocks of variables, solving a tiny implicit problem per block and applying each update immediately: one $3\times3$ Newton solve per vertex, with vertices graph-colored so that a color class updates in parallel. For any such solver the Jacobian of the sweep is an ordered product of block-update Jacobians, so its transpose is that product in reverse. The adjoint is therefore *the same sweep, backwards*, with each block update replaced by its local transpose: same coloring, same constant block size, same memory layout, and no global system at any point. We read this as a structural duality (forward block sweep $\Leftrightarrow$ reverse adjoint sweep) with a thesis attached: *equation-level differentiation destroys the solver locality that block-coordinate methods are built around; solver-level differentiation preserves it*. The backward pass is a solver of the same shape as the forward, and everything that made the forward fast makes the backward fast.

**What we are and are not claiming.** We do not claim custom gradients are novel: a custom vector–Jacobian product can always be handed to an AD framework. But that product still has to *compute* something, and an implicit step admits two computations: (a) assemble and solve a global adjoint system, as every prior adjoint-based differentiable elastodynamics method does, or (b) run a reverse sweep of constant-size local solves and never form such an object. Route (b) removes the memory wall; we contribute its construction, exactness conditions, proof, and GPU implementation, plus the measurement that (a) and (b) do not merely differ in cost: at finite solver depth they return *different gradients*, and only (b) differentiates what was executed.

Exactness at finite depth is the subtle part. Three terms are required, and dropping any one yields gradients that look correct at convergence and are badly wrong before it: the local adjoint solve, the Hessian-tangent term, and, easily overlooked, the Jacobians of non-smooth safeguards (update saturation, regularization floors, validity gates). We found all three in our own implementation against an autograd reference, and report their error laws because they are invisible to the usual finite-difference-at-converged-$K$ protocol.

**Contributions.**

- The exact discrete adjoint of an executed block-coordinate implicit sweep (§3): a reverse sweep of constant-size local adjoint solves that forms no global linear-algebra object, with the structural conditions under which it is exact (Theorem 1).
- An analysis of what exactness at *finite* solver depth requires (§3.5), including non-smooth safeguard terms that prior hand-derived adjoints omit, with the measured error law of each omission (§6.2).
- A controlled comparison (one codebase, one GPU, identical meshes, and forward trajectories verified identical to $10^{-16}$) against unrolled AD and against both assembled and matrix-free equation-level adjoints (§6.3); plus inverse-problem tasks, machine-precision transfer to projective and extended position-based dynamics (App. B), and differentiable elastodynamics at $10^6$ contact-coupled bodies (8M vertices).

**Scope.** We implement and evaluate a single solver family, with Vertex Block Descent as the concrete instantiation. Theorem 1 states the structural assumptions under which the construction transfers; minimal instantiations on extended position-based dynamics (Macklin et al., 2016) and projective dynamics (Bouaziz et al., 2014) confirm the transfer, exact to machine precision at every tested depth (App. B).

## 2 Background and notation

**Notation.** Bold italic denotes vectors ($\boldsymbol{x}$, $\boldsymbol{v}$, $\boldsymbol{y}$, $\boldsymbol{g}_i$, $\boldsymbol{p}_i$, $\Delta\boldsymbol{x}_i$); roman capitals denote matrices ($M$, $H_i$, $I$); plain italic denotes scalars ($h$, $K$, $\mu$, $\lambda$, $\epsilon$, and indices $i, j, t$); italic capitals denote operators

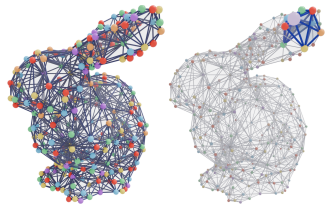

**Figure 1:** Vertex Block Descent on a tetrahedral mesh. Left: distance-1 graph coloring; vertices that share an element never share a color, so each color class updates in parallel. Right: one local 3×3 Newton solve, touching only the elements incident to the updated vertex.

on state $(S_i, U_i, F_i)$. An overbar denotes an *adjoint*: for a downstream scalar loss $L$, $\bar{\boldsymbol{x}} \coloneqq \partial L/\partial \boldsymbol{x}$. We write $|V|$ for the vertex count, so a mesh has $3|V|$ degrees of freedom (DoF), and $|T|$ for the tetrahedron count. **Solver depth** $K$ is the number of inner sweeps executed per time step, where one sweep visits every vertex once in color order; $K$ is a fixed user-chosen budget, not a convergence tolerance, and small $K$ is the common regime because it is fast.

**Vertex Block Descent.** Chen et al. (2024a) solve Eq. (1) by block coordinate descent on the variational form of implicit Euler, taking one vertex position as a block and sweeping the blocks in Gauss–Seidel order. Each local solve targets the same global objective, and the scheme is reported unconditionally stable and, unlike a truncated global Newton solve, *remains* stable when the iteration count is capped: the sweep count is a compute budget rather than a tolerance, so finite $K$ is the normal operating point, and exactly the regime where an equation-level adjoint stops describing what ran. Concretely, for vertex $i$ with lumped mass $m_i$ and incident elements $\mathcal{E}(i)$,

$$\begin{aligned} \boldsymbol{g}_i(\boldsymbol{x}) &= \tfrac{m_i}{h^2}(\boldsymbol{x}_i - \boldsymbol{y}_i) + \sum_{e\in\mathcal{E}(i)} \nabla_{\boldsymbol{x}_i} E_e(\boldsymbol{x}), \\ H_i(\boldsymbol{x}) &= \tfrac{m_i}{h^2} I_3 + \sum_{e\in\mathcal{E}(i)} \nabla^2_{\boldsymbol{x}_i\boldsymbol{x}_i} E_e(\boldsymbol{x}), \end{aligned} \tag{2}$$

and the local Newton step is applied immediately (Gauss–Seidel):

$$H_i(\boldsymbol{x})\,\Delta\boldsymbol{x}_i = -\boldsymbol{g}_i(\boldsymbol{x}), \qquad \boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \Delta\boldsymbol{x}_i. \tag{3}$$

Vertices are distance-1 graph-colored, so vertices sharing an element never share a color and each color class updates in parallel (Fig. 1). Vertex Block Descent is thus a block implicit solver whose fundamental operation is a constant-size 3×3 solve. Elastic energy is stable Neo-Hookean (Smith et al., 2018) throughout.

## 3 The Adjoint of the Executed Sweep

### 3.1 Block Updates and the Solver-Induced Inverse

Write the executed solver as a composition of block updates $S = S_n \circ S_{n-1} \circ \cdots \circ S_1$, where $S_i$ modifies only block $i$:

$$\boldsymbol{x}_i^{+} = U_i(\boldsymbol{x}) = \boldsymbol{x}_i + \Delta\boldsymbol{x}_i(\boldsymbol{x}), \qquad F_i(\Delta\boldsymbol{x}_i, \boldsymbol{x}) = \boldsymbol{0}, \tag{4}$$

with $\Delta\boldsymbol{x}_i$ defined implicitly by $F_i$; for Vertex Block Descent, $F_i(\Delta\boldsymbol{x}_i, \boldsymbol{x}) = H_i(\boldsymbol{x})\Delta\boldsymbol{x}_i + \boldsymbol{g}_i(\boldsymbol{x})$. Iterating this composition realizes an *approximate inverse* of the global stationary system: $S$ maps the predictor $\boldsymbol{y}$ to a state of lower $G$-energy, and finitely many sweeps reduce the residual. We call $S$ the solver-induced inverse; it is block-structured by construction.

### 3.2 Transpose Action, and Contrast with Equation-Level Differentiation

The Jacobian of the sweep is the ordered product of block-update Jacobians, so

$$(\partial S)^\top = (\partial S_1)^\top (\partial S_2)^\top \cdots (\partial S_n)^\top. \tag{5}$$

Two consequences. Reverse ordering is the computational form of the transpose of a composition, not a design choice. And because each $\partial S_i$ touches only one block, so does each $(\partial S_i)^\top$: the backward is the transpose of the forward sweep block by block, with the same color partition in reverse and the same constant block size.

This differentiates a different object from equation-level implicit differentiation, which replaces the executed solver by the converged equation and inverts $\nabla^2 G(\boldsymbol{x}^\star)$:

$$\underbrace{(\partial S)^\top}_{\substack{\text{adjoint of the finite executed}\\ \text{solver; block-local}}} \qquad \text{vs.} \qquad \underbrace{\left(\nabla^2 G(\boldsymbol{x}^\star)\right)^{-\top}}_{\substack{\text{adjoint of the converged}\\ \text{system; global solve}}} \tag{6}$$

The two coincide only in the convergence limit. §6.1 measures the gap and shows it is large in the regime practitioners actually use.

### 3.3 Local adjoint of one block update

Differentiating $F_i = \mathbf{0}$ with respect to an arbitrary variable $z$ gives $\frac{\partial F_i}{\partial \Delta \boldsymbol{x}_i}\frac{d\Delta \boldsymbol{x}_i}{dz} + \frac{\partial F_i}{\partial z} = \mathbf{0}$, and since $\partial F_i/\partial \Delta \boldsymbol{x}_i = H_i$,

$$\frac{d\Delta \boldsymbol{x}_i}{dz} = -H_i^{-1}\left(\frac{\partial \boldsymbol{g}_i}{\partial z} + \frac{\partial H_i}{\partial z}\,\Delta \boldsymbol{x}_i\right). \tag{7}$$

We differentiate the local *optimality condition*, not the implementation of the local linear solver: the backward is organized by the implicit relation itself rather than by an expanded reverse-mode graph.

Let $\bar{\boldsymbol{x}}_i^+$ be the *incoming* adjoint at block $i$, the one accumulated by every reverse update that has already executed, and $\bar{\boldsymbol{x}}_i^-$ the *outgoing* adjoint passed further back (Fig. 2). Because $\boldsymbol{x}_i^+ = \boldsymbol{x}_i^- + \Delta \boldsymbol{x}_i(\boldsymbol{x})$, the chain rule splits into a direct identity branch and an implicit branch:

$$\bar{\boldsymbol{x}}_i^- \;=\; \bar{\boldsymbol{x}}_i^+ \;+\; \left(\tfrac{d\Delta \boldsymbol{x}_i}{d\boldsymbol{x}_i}\right)^\top \bar{\boldsymbol{x}}_i^+. \tag{8}$$

Introducing the local adjoint variable $\boldsymbol{p}_i$ defined by

$$H_i^\top \boldsymbol{p}_i = \bar{\boldsymbol{x}}_i^+, \tag{9}$$

and combining with Eq. (7), the implicit branch for *any* upstream variable $z$ becomes

$$\left(\tfrac{d\Delta \boldsymbol{x}_i}{dz}\right)^\top \bar{\boldsymbol{x}}_i^+ = -\left(\tfrac{\partial \boldsymbol{g}_i}{\partial z}\right)^\top \boldsymbol{p}_i \;-\; \left(\tfrac{\partial H_i}{\partial z}\right) : \left(\boldsymbol{p}_i\, \Delta \boldsymbol{x}_i^\top\right). \tag{10}$$

Here $A : B = \sum_{a,b} A_{ab} B_{ab}$, so the second term is the contraction of the third-order tensor $\partial H_i/\partial z$ against the rank-one matrix $\boldsymbol{p}_i \Delta \boldsymbol{x}_i^\top$,

$$\left(\tfrac{\partial H_i}{\partial z}\right) : \left(\boldsymbol{p}_i \Delta \boldsymbol{x}_i^\top\right) \;=\; \sum_{a,b=1}^{3} \frac{\partial [H_i]_{ab}}{\partial z}\,[\Delta \boldsymbol{x}_i]_b\,[\boldsymbol{p}_i]_a. \tag{11}$$

No third-order tensor is ever formed: Eq. (11) is evaluated as a directional derivative of $H_i$ along $\Delta \boldsymbol{x}_i$, tested against $\boldsymbol{p}_i$, at the cost of a few $3\times3$ products.

Setting $z = \boldsymbol{x}_i$ recovers the implicit summand of Eq. (8); $z = \boldsymbol{x}_j$ accumulates the off-diagonal contribution into $\bar{\boldsymbol{x}}_j^-$. For a material parameter $\theta \in \{\mu, \lambda\}$, or any other upstream quantity the block depends on (nodal forces, rest shape), the same expression accumulates

$$\bar{\theta} \;\mathrel{+}=\; -\left(\tfrac{\partial \boldsymbol{g}_i}{\partial \theta}\right)^\top \boldsymbol{p}_i \;-\; \left(\tfrac{\partial H_i}{\partial \theta}\right) : \left(\boldsymbol{p}_i\, \Delta \boldsymbol{x}_i^\top\right), \tag{12}$$

summed over every block update in which $\theta$ participated. State and parameter adjoints therefore share one local solve $H_i^\top \boldsymbol{p}_i = \bar{\boldsymbol{x}}_i^+$; parameters cost only the extra contractions, which is why adding material gradients to a rollout does not change the asymptotic cost of the backward. Together Eqs. (8)–(10) are the complete local vector–Jacobian product of one block update. We call the second term of Eq. (10) the **Hessian tangent** ("stage 2"), since it exists only because $H_i$ itself depends on state; the first is the gradient tangent ("stage 1"). Every quantity is $3\times3$; nothing global is formed.

### 3.4 Reverse-sweep adjoint principle

**Theorem 1** (Exact discrete adjoint of a block implicit solver)**.** *Let $S = S_n \circ \cdots \circ S_1$ be a finite composition of block implicit updates of the form* (4)*. Assume each $F_i$ is $C^1$ in a neighborhood of its executed linearization state and that $\partial F_i/\partial \Delta \boldsymbol{x}_i$ is invertible there; for the affine instance $F_i = H_i \Delta \boldsymbol{x}_i + \boldsymbol{g}_i$ it suffices that $H_i$ and $\boldsymbol{g}_i$ are differentiable at that state with $H_i$ invertible. Then*

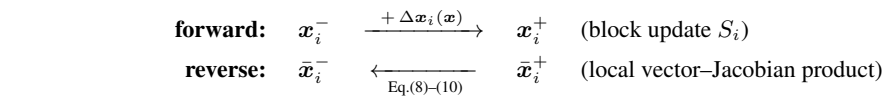


**Figure 2:** Data flow around one block update. The forward advances the block state; the reverse consumes the *incoming* adjoint $\bar{\boldsymbol{x}}_i^+$ (accumulated by reverse updates already executed) and produces the *outgoing* adjoint $\bar{\boldsymbol{x}}_i^-$. Superscripts denote position in the sweep, not before/after in time.

$(\partial S)^\top = (\partial S_1)^\top \cdots (\partial S_n)^\top$*, with each* $(\partial S_i)^\top$ *realized by the local vector–Jacobian product of Eqs.* (8)–(10)*: a direct identity contribution, a local adjoint solve* $H_i^\top \boldsymbol{p}_i = \bar{\boldsymbol{x}}_i^+$*, and a local scatter. Provided the backward replays the same local linearization states as the forward, the adjoint of one sweep is a reverse block sweep of constant-size local solves.*

The proof is in Appendix A. The qualifier that matters: this is exact for the *finite executed* $S$, not merely in the convergence limit. It does not assert that the block factorization is unique or that it converges to the global linearized inverse for any particular $K$.

### 3.5 What exactness at finite depth actually requires

Theorem 1 assumes each $F_i$ is differentiated in full. Production block solvers contain safeguards that are easy to treat as invisible, and each omission produces gradients that are wrong at small $K$ and self-correcting at large $K$, so they hide from convergence-regime testing. We isolate three, all of which we found in our own implementation.

**(i) Update saturation.** Implementations commonly damp the raw step, $\Delta\boldsymbol{x} \leftarrow \Delta\boldsymbol{x}\, c/(\|\Delta\boldsymbol{x}\| + c + \varepsilon)$ with damping constant $c$ (App. H gives the values), to stabilize early sweeps. Its Jacobian is a rank-one correction that must be applied to the incoming adjoint *before* the local solve:

$$J = s\, I - \tfrac{1}{c\, r}\, \Delta\boldsymbol{x}\, \Delta\boldsymbol{x}^\top, \qquad r = \|\Delta\boldsymbol{x}^{\text{raw}}\|, \quad s = \|\Delta\boldsymbol{x}\|/r. \tag{13}$$

Omitting it gives relative error $\propto \|\Delta\boldsymbol{x}\|/(\|\Delta\boldsymbol{x}\| + c)$, measured up to $26\%$ at $K{=}1$ in Table 2, vanishing as $\|\Delta\boldsymbol{x}\| \to 0$.

**(ii) The Hessian tangent.** The second term of Eq. (10) is often dropped from the *state* adjoint and retained, if at all, only for parameter gradients. For the block Hessian of Eq. (2) the state dependence enters only through $\boldsymbol{q}_i = \operatorname{cof}(F)\nabla N_i$, so the term reduces to two cross products per (element, vertex) pair. Omitting it leaves error $\propto \|\Delta\boldsymbol{x}\|$.

**(iii) Validity gates.** Tangent terms are sometimes disabled on compressed elements (e.g. applied only when $\det F > \tau$) while the forward applies no such cutoff. This is the most damaging of the three: its effect is non-local and *grows with mesh refinement*: one gated element out of $245$ moves the whole gradient by $8.2 \times 10^{-3}$, and the gated terms contain no division by $\det F$, so the gate buys no safety.

None of the three is visible from finite-difference checks at converged $K$, which is the standard validation protocol; all three are visible immediately against an autograd reference at $K{=}1$ (§6.2).

## 4 Differentiable Vertex Block Descent

We instantiate the principle on Vertex Block Descent. Each forward update records its local displacement $\Delta\boldsymbol{x}_i$ and Hessian block; the backward traverses iterations and colors in reverse, undoing the recorded displacement to reconstruct the linearization point, applying Eq. (13), passing the direct adjoint contribution, solving $H_i^\top \boldsymbol{p}_i = \bar{\boldsymbol{x}}_i^+$, and scattering the stage-1 and stage-2 contributions to neighboring vertices and material parameters via Eq. (10). The same backward yields gradients with respect to initial state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$, Lamé parameters $(\mu, \lambda)$, time-varying external forces $\boldsymbol{f}_t$, and rest-shape positions, in one reverse sweep with no global assembly and no graph rebuild between parameter sets (closed forms: App. C). Forward and backward share the color partition, neighborhood structure, memory layout, and thread schedule, so the backward inherits the forward's parallelism directly (Algorithm 1). Relative to unrolled AD the difference is what is *not* stored: the tape collapses to a per-sweep replay buffer of nine doubles per active vertex – the applied update and the six entries of the symmetric block, 72 bytes.

**Algorithm 1** Forward Vertex Block Descent and its reverse adjoint

**for** $k = 0, \dots, K-1$ **do** ▷ forward
  **for** color $c = 1, \dots, C$ **do**
    **for all** vertices $i$ in color $c$ **in parallel do**
      assemble $\boldsymbol{g}_i, H_i$; solve $H_i \Delta \boldsymbol{x}_i = -\boldsymbol{g}_i$; saturate
      record $\Delta \boldsymbol{x}_i, H_i$; $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \Delta \boldsymbol{x}_i$
    **end for**
  **end for**
**end for**
**for** $k = K-1, \dots, 0$ **do** ▷ reverse
  **for** color $c = C, \dots, 1$ **do**
    **for all** vertices $i$ in color $c$ **in parallel do**
      undo $\Delta \boldsymbol{x}_i$; apply saturation Jacobian (13) to $\bar{\boldsymbol{x}}_i^+$
      solve $H_i^\top \boldsymbol{p}_i = \bar{\boldsymbol{x}}_i^+$
      scatter stage-1 and stage-2 terms (10) to neighbors / parameters
    **end for**
  **end for**
**end for**

## 5 Implementation

We provide two GPU backends, NVIDIA Warp (Macklin, 2022) kernels and a fused CUDA C++ path compiled on demand, differing by 7–22× in runtime through kernel fusion but agreeing to $10^{-16}$ on forward state and gradients. Memory grows with replay information, not with an AD graph, and CUDA graph capture removes launch overhead in outer loops. The per-vertex block defaults to the positive-definite approximation $H_i = \mu \|\nabla N_i\|^2 I + \lambda \boldsymbol{q}\boldsymbol{q}^\top$ ($\nabla N_i$ the constant shape-function gradient) in the style of position-based nonlinear Gauss–Seidel (Chen et al., 2024c); an exact $9\times9$-tangent variant covers the full indefinite block of Chen et al. (2024a).

**Contact.** Two primitives coexist, and both are differentiated the same way as the elastic term, by the reverse sweep rather than by a separate mechanism. The first is the incremental potential contact (IPC) log barrier (Li et al., 2020) against the floor and against the vertex's own neighborhood, $b(d) = -\kappa\,(d - \hat{d})^2 \log(d/\hat{d})$ for gap $d < \hat{d}$, with lagged dissipation applied only to downward motion. Because the barrier contributes to the same per-vertex $\boldsymbol{g}_i$ and $H_i$ of Eq. (2), its derivative enters the backward through the existing local solve, and no contact-specific adjoint is added; the barrier's second derivative sits in $H_i$, and its third derivative is carried by the Hessian-tangent term of Eq. (10). Adjoints therefore flow through every barrier evaluation of a rollout, not through a converged contact state. The second primitive is a differentiable center-of-mass penalty between separate bodies, with a uniform-grid broad phase (cell keys sorted once, a lower/upper-bound binary-search pair per neighbor cell) that stays near-linear at $10^6$ bodies, used when the object count makes vertex-level self-collision impractical. Pair indices carry no gradient, so the search algorithm never touches differentiability.

This is the substantive difference from DiffIPC (Huang et al., 2024), which differentiates a converged incremental potential and therefore inherits a global step Hessian on the backward; ours differentiates the executed barrier evaluations and stays block-local, which is why the memory figures quoted in §6.4 differ by orders of magnitude. Symmetrically, where the contact set is still changing, our gradient reflects the executed sweep, not a converged contact configuration. Both backends expose a tensor-native interface, so backpropagation through time runs GPU-resident. Appendix H gives details.

## 6 Experiments

Optimization tasks use Adam (Kingma & Ba, 2015). §6.1–§6.5 run on a single RTX 4090 Laptop GPU (16 GB, CUDA 12.1, PyTorch 2.3), §6.4's $K \times |V|$ grid on an RTX A6000, and §6.7 on one H100 (80 GB). Every backward route in §6.1–§6.3 is measured on the *same* GPU within a single process, so no comparison below crosses hardware.

**Table 1: Gradient error vs. the autograd reference** (relative, max over $\bar{\boldsymbol{x}}_0, \bar{\boldsymbol{v}}_0$) as solver depth $K$ varies. Our reverse sweep is exact at every depth. The equation-level adjoint differentiates a converged solve, so it is accurate only once the executed state satisfies $\nabla G = \mathbf{0}$; the residual column quantifies how badly that assumption fails at small $K$. Mesh $|V|$=115 (345 DoF), backpropagation through four rollout steps.

| $K$ | $\|\nabla G(\boldsymbol{x}^\star)\|$ | Ours | Global adj. (assembled) | Global adj. (matrix-free) |
|---|---|---|---|---|
| 1 | $1.6 \times 10^{-1}$ | $\mathbf{3.6 \times 10^{-15}}$ | $3.7 \times 10^{-1}$ | $3.7 \times 10^{-1}$ |
| 5 | $4.7 \times 10^{-9}$ | $\mathbf{3.3 \times 10^{-15}}$ | $2.3 \times 10^{-5}$ | $2.3 \times 10^{-5}$ |
| 20 | $1.0 \times 10^{-14}$ | $\mathbf{2.1 \times 10^{-15}}$ | $9.7 \times 10^{-10}$ | $1.6 \times 10^{-10}$ |

**Table 2: Ablation of §3.5**: relative gradient error vs. the reference at $K$=1, across mesh refinement.

| Variant | $\|V\|$=26 | $\|V\|$=115 | $\|V\|$=631 |
|---|---|---|---|
| Full adjoint (ours) | $\mathbf{4.2 \times 10^{-16}}$ | $\mathbf{5.1 \times 10^{-16}}$ | $\mathbf{9.1 \times 10^{-16}}$ |
| w/o saturation Jacobian (i) | $7.7 \times 10^{-2}$ | $1.6 \times 10^{-1}$ | $2.6 \times 10^{-1}$ |
| w/o Hessian tangent (ii) | $2.9 \times 10^{-4}$ | $8.2 \times 10^{-3}$ | $1.1 \times 10^{-1}$ |
| with validity gate $\det F > 0.5$ (iii) | $4.2 \times 10^{-16}$ | $8.2 \times 10^{-3}$ | $1.2 \times 10^{-1}$ |

**Baselines and the reference gradient.** We compare four backward routes over an *identical* forward: (1) **ours**; (2) **unrolled AD**, PyTorch autograd through the executed colored sweep; (3) **global adjoint (assembled)**, building the sparse $3|V| \times 3|V|$ Hessian and solving the adjoint system by preconditioned conjugate gradients (PCG), the equation-level route of DiffPD, DiffCloth, DiffQN and DiffIPC (theirs with prefactored or direct solvers); (4) **global adjoint (matrix-free)**, never forming the Hessian, applying it via double-backward Hessian-vector products inside PCG. We include (4) deliberately: it is the leanest possible opponent on memory, and is exactly the design point of supplying a custom vector–Jacobian product for the linear solve inside an AD framework. Routes (3) and (4) agree to $2 \times 10^{-16}$, certifying they solve the same system.

Unrolled AD serves twice. As a competitor it shows the cost of a generic tape. As a *reference* it is exact by construction, since autograd through the executed program is, to floating point, the derivative of what ran, so all gradient errors below are relative errors against it. We verify that all backends produce the same forward trajectory to $10^{-16}$ before comparing any gradient.

### 6.1 Exactness at finite solver depth

Table 1 is the central measurement. At $K$=1 the executed state violates the stationary equation by $1.6 \times 10^{-1}$ and the equation-level gradient is wrong by 37%; ours is exact. By $K$=20 the solver has converged and the routes agree, as Theorem 1 predicts. This is the operational content of differentiating the solver rather than the equation: the two are not two implementations of one gradient but gradients of two different functions, and only one of them is the function the forward pass evaluated. Exactness holds across every axis swept (meshes $|V| \in \{26, 115, 631\}$, $K \in \{1, 5, 20\}$, one- and four-step rollouts, three decades of stiffness, two step sizes, excitation scale over two decades): worst case $3.6 \times 10^{-15}$, on both GPU backends (finite-difference validation: App. F).

### 6.2 Ablation: the three exactness terms

Each omission in Table 2 is invisible at convergence, severe before it, and *grows* under refinement. The variants are earlier revisions of our own implementation; the gated variant (iii) is exact on the coarse mesh only because no element there approaches inversion.

### 6.3 Cost: a controlled comparison

Table 3 gives 33× less time and 69× less memory than unrolled AD, and the regimes separate cleanly: on a well-conditioned step ($h$=$10^{-2}$, soft material) PCG converges in ∼5 iterations per rollout step and the equation-level route is at its closest, while at $\mu$=$\lambda$=$5 \times 10^5$ PCG needs 233 iterations in total and the equation-level backward rises to 254–391 ms; ours does not move with stiffness, because a reverse sweep never solves a global system whose difficulty tracks the material.

**Table 3: Backward cost: same codebase, GPU, and mesh**, with forward trajectories verified identical to $10^{-16}$, so no part of the gap is attributable to a faster forward, a different mesh, or different hardware. $|V|$=115, $K$=20, backpropagation through four rollout steps. "PCG" is the conjugate-gradient iteration count. Peak memory is the caching-allocator peak (`torch.cuda.max_memory_allocated`).

| Backward route | Time (ms) | Peak memory (MB) | PCG iters | Gradient error |
|---|---|---|---|---|
| Ours | **28.5** | **0.9** | n/a | $2.1 \times 10^{-15}$ |
| Unrolled AD (reference) | 939.9 | 64.3 | n/a | n/a |
| Global adjoint (assembled) | 257.6 | 19.8 | 36 | $9.7 \times 10^{-10}$ |
| Global adjoint (matrix-free) | 89.4 | 17.7 | 40 | $1.6 \times 10^{-10}$ |

**Table 4:** Inverse-problem results, all using the DiffVBD adjoint with Adam. **#vars** is the number of scalar optimization variables. **Final loss** is the converged objective in task units: relative parameter-recovery loss (dimensionless) for material ID; per-vertex squared position error ($m^2$) for dress and silhouette; squared trajectory error ($m^2$) for wind; squared center-of-mass-to-target distance ($m^2$) for the bounces. **Outcome** is task-interpretable: relative material error; for wind, $\|\Delta w\|$ is the $L_2$ deviation of the recovered piecewise-constant wind from the reference, marked under-determined for the reason given in §6.5; for the bounces, the final center-of-mass-to-target distance, against trajectory envelopes of ≈1.8 m (cube) and ≈0.8 m (bunny), so 0.013 m and 0.034 m are small relative to the motion.

| Task | #vars | Iters | Final loss | Outcome |
|---|---|---|---|---|
| Beam $\mu$ ID | 1 | 76 | $<10^{-4}$ | $\mu$: 0.1% error |
| Beam $\mu$, $\lambda$ ID | 2 | 100 | $2.0 \times 10^{-4}$ | $\mu$, $\lambda$: <0.25% |
| Dress $\mu$, $\lambda$ ID | 2 | 250+5 | $4.8 \times 10^{-9}$ $m^2$ | $\mu$: 0.01%, $\lambda$: 0.03% |
| Dress silhouette | 2 | 350+5 | $<10^{-12}$ $m^2$ | $\mu$: <0.01%, $\lambda$: <0.01% |
| Flag wind control | 24 | 199 | $5.1 \times 10^{-3}$ $m^2$ | $\|\Delta w\|$=15.3 (under-det.) |
| Cube bounce | 6 | 151 | $1.6 \times 10^{-2}$ $m^2$ | center-of-mass–target 0.30 → 0.013 m |
| Bunny bounce | 3 | 100 | $3.1 \times 10^{-2}$ $m^2$ | center-of-mass–target 0.035 → 0.034 m |
| Cover the Spot | 18 | 600 | n/a | uncovered area 0.0% |

### 6.4 Runtime and scaling

The backward-to-forward ratio is *not* a single number: at small $|V|$ it is dominated by fixed per-color kernel-launch overhead, and as $|V|$ grows the ratio approaches 1, measuring 0.96–1.14 across the full $K \times |V|$ grid up to $|V|$=201k, where the unrolled tape no longer fits in device memory (App. K, with per-backend timings). Memory is linear in $K|V|$ at ≈4 KB per vertex per sweep (±13% across the grid). For reference, DiffIPC (Huang et al., 2024) reports ∼100 GB at $|V|$=9,939 against our 1,074 MB at $|V|$=27,169; a cross-paper comparison of published numbers, unlike Table 3. Solver depth also affects what an optimizer achieves, not only gradient accuracy; on our small initial-state task the effect does not separate from run-to-run spread (null result in App. G).

### 6.5 Inverse problems

We verify the gradients are useful, not merely correct. Each task states what is given, what is optimized, and the success criterion; Table 4 reports variables, iterations, final loss with units, and outcome (narratives App. I, protocols App. J, more tasks App. E).

### 6.6 Real-to-sim: material identification from captured video

We fit materials of real deformables from PhysTwin's RGB-D captures (Jiang et al., 2025): tracked 3D points give per-frame correspondence targets, recorded grasp trajectories drive pinned vertices as moving Dirichlet boundaries, and the frame-0 shape prior is tetrahedralized. Global $(\mu, \lambda)$ are fitted by backpropagation through time on each capture's train split and evaluated held-out (Table 10; details in App. L). On a stuffed sloth stretched by both arms, a deliberately too-soft start improves mean track error 6.6 → 1.9 cm (train) and 13.1 → 2.4 cm (held-out); a doubly lifted cloth improves to 0.5 cm, with two implicit steps per frame versus 667 explicit substeps. Train-selected corrections (App. L) bring the sloth to 1.01/1.44 cm, its train error matching the dataset authors' released per-scene fit. The grasp bounds compliance only from below (App. L).

### 6.7 SCALE

We optimize per-body launch velocities $\boldsymbol{v}_0$ for $10^6$ tetrahedralized cubes (edge 3 mm, $\sim$8M vertices, $\sim$6M tetrahedra packed as one mega-mesh) so that at $t^\star{=}0.6$ s each body's center of mass reaches a target sampling a ghost-emoji silhouette; bodies continue ballistically past $t^\star$, so the optimizer must place each on a trajectory whose apex crosses its target at exactly that step. Rollouts are $240$ implicit steps ($h{=}5$ ms, $K{=}5$) with backpropagation through time, gradient-checkpointed (Chen et al., 2016) every $5$ steps under a Gaussian-time-windowed quadratic objective. Bodies interact through the differentiable center-of-mass contact penalty of §5 (uniform-grid broad phase), and the backward differentiates through every contact impulse. A contact-free stage brings bodies near their targets ($16.0$ h, $\sim$71 GB, mean error $0.562$ m); $200$ Adam iterations under the contact-enabled dynamics land at $0.565$ m ($16.7$ h, $\sim$74 GB on one H100), suppressing interpenetration (overlapping pairs halve, from $0.088$ to $0.042$ per body) at a cost of $\sim$3.5 mm; the loss falls $1.09 \times 10^6 \to 3.92 \times 10^5$ ($64\%$). The experiment stress-tests differentiability under simultaneous solver-depth, object-count, contact, and memory pressure.

## 7 RELATED WORK

**Differentiable simulators.** Differentiable physics is established for soft bodies (Hu et al., 2020; 2019; Jatavallabhula et al., 2021; Qiao et al., 2020), cloth and contact (Liang et al., 2019; Geilinger et al., 2020; Li et al., 2023), articulated bodies (Werling et al., 2021), and Eulerian PDEs (Holl et al., 2020); implementations either unroll a fixed-iteration stepper through reverse-mode AD or differentiate the converged equation analytically: both dissolve the forward's block structure.

**Discrete adjoints of iterative solvers.** Differentiating the stationary equation via the implicit function theorem (Giles & Pierce, 2000; Hinze et al., 2009; Blondel et al., 2022) underlies deep equilibrium models (Bai et al., 2019), optimization layers (Amos & Kolter, 2017), and adjoint multigrid for steady-state PDEs (Giles, 2000), with its continuous-time counterpart in Neural ODEs (Chen et al., 2018). That literature classically distinguishes differentiating a converged equation from a finite iterative process; we carry the second option down to the granularity of a block-coordinate sweep, where it becomes matrix-free. In elastodynamics, DiffPD and DiffCloth (Du et al., 2022; Li et al., 2023), DiffQN (Cai et al., 2025), and DiffIPC (Huang et al., 2024) all place a global linear-algebra object on the backward path (Table 5); DiffXPBD (Stuyck & Chen, 2023) gives analytic gradients of position-based constraint projections; the nearest block-implicit neighbor outside graphics takes a global conjugate-gradient backward around a Gauss–Seidel forward (Chen et al., 2026). Operator-level analytic adjoints are long-standing in differentiable fluid control (McNamara et al., 2004; Tang et al., 2021; Chen et al., 2024b); we push that granularity from a per-step global operator to the per-vertex $3{\times}3$ block, so the backward inherits the forward's parallel schedule.

**Structured solver updates.** A broad family of simulation methods exploits local or block-structured updates, including the block-coordinate mass–spring solver of Liu et al. (2013), the local–global iterations of projective dynamics (Bouaziz et al., 2014), the constraint-projection updates of PBD and XPBD (Müller et al., 2007; Macklin et al., 2016), and Vertex Block Descent (Chen et al., 2024a). These methods admit implementations that exploit data locality and, where update conflicts permit, graph-coloring-based parallelism. Our contribution is a backward pass that preserves this computational organization.

## 8 DISCUSSION AND LIMITATIONS

Equation-level differentiation discards the locality block implicit solvers were designed to exploit; the solver-level alternative recovers it: a forward that scales to millions of vertices becomes differentiable for one forward-equivalent reverse pass and a per-sweep replay buffer. **Limitations.** *(i)* the full evaluation covers one solver family; projective and extended position-based dynamics are verified on minimal systems (App. B); *(ii)* a fully *converged* solve's derivative is better served equation-level. *(iii)* Backward replay uses atomic-add scatter, and Coulomb friction with active-set changes is future work.

### Reproducibility statement

All claims in §6.1–§6.3 are produced by the harness released with this submission, which constructs every scene procedurally (no data dependencies), verifies forward-trajectory agreement across backends before comparing gradients, and writes machine-readable results. The exactness comparison, the ablation of §3.5, and the cost table are each a single command. Exact commands, hardware, and library versions are in Appendix M; Theorem 1 is proved in Appendix A.

### The use of large language models

LLM-based assistants were used during this project to draft and polish portions of the text under the authors' direction, to help execute and monitor experiments, and for literature retrieval and discovery. The algorithms and their implementations are entirely the authors' own work, as are all proofs and experimental designs; all quantitative results were produced by the released code and verified by the authors, and all LLM-assisted text was reviewed and edited by the authors.

### Ethics statement

This work concerns numerical methods for physical simulation and raises no human-subjects, privacy, or fairness concerns known to us. Differentiable simulation carries the dual-use potential common to physical modeling generally; we identify no application-specific risk introduced here.

## References

Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning (ICML)*, 2017.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of SIGGRAPH*, 1998.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Transactions on Graphics*, 33 (4):154:1–154:11, 2014.

Youshuai Cai, Chen Li, Haichuan Song, Youchen Xie, and ChangBo Wang. DiffQN: Differentiable quasi-newton method for elastodynamics. In *Pacific Graphics Conference Papers, Posters, and Demos*. The Eurographics Association, 2025. ISBN 978-3-03868-295-0. doi: 10.2312/pg.20251265.

Anka He Chen, Ziheng Liu, Yin Yang, and Cem Yuksel. Vertex block descent. *ACM Transactions on Graphics*, 43(4):116:1–116:16, 2024a. doi: 10.1145/3658179.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Siyuan Chen, Minghao Guo, Caoliwen Wang, Anka He Chen, Yikun Zhang, Jingjing Chai, Yin Yang, Wojciech Matusik, and Peter Yichen Chen. Physically valid biomolecular interaction modeling with Gauss–Seidel projection. In *International Conference on Learning Representations (ICLR)*, 2026.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Yixin Chen, David I. W. Levin, and Timothy R. Langlois. Fluid control with Laplacian eigenfunctions. In *ACM SIGGRAPH 2024 Conference Papers*, 2024b.

Yizhou Chen, Yushan Han, Jingyu Chen, Zhan Zhang, Alex McAdams, and Joseph Teran. Position-based nonlinear Gauss-Seidel for quasistatic hyperelasticity. *ACM Transactions on Graphics*, 43 (4):115:1–115:15, 2024c. doi: 10.1145/3658154.

Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis Wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13:6, 2019.

Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. DiffPD: Differentiable projective dynamics. *ACM Transactions on Graphics*, 41(2): 13:1–13:21, 2022. doi: 10.1145/3490168.

Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. ADD: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics*, 39(6):190:1–190:15, 2020. doi: 10.1145/3414685.3417766.

Michael B. Giles. On the use of Runge-Kutta time-marching and multigrid for the solution of steady adjoint equations. Technical Report NA-00/10, Oxford University Computing Laboratory, 2000.

Michael B. Giles and Niles A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, 65(3–4):393–415, 2000.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2 edition, 2008.

Michael Hinze, René Pinnau, Michael Ulbrich, and Stefan Ulbrich. *Optimization with PDE Constraints*, volume 23 of *Mathematical Modelling: Theory and Applications*. Springer, 2009. ISBN 978-1-4020-8838-4.

Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control PDEs with differentiable physics. In *International Conference on Learning Representations (ICLR)*, 2020.

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

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations (ICLR)*, 2020.

Zizhou Huang, Davi Colli Tozoni, Arvi Gjoka, Zachary Ferguson, Teseo Schneider, Daniele Panozzo, and Denis Zorin. Differentiable solver for time-dependent deformation problems with contact. *ACM Transactions on Graphics*, 43(3):31:1–31:30, 2024. doi: 10.1145/3657648.

Krishna Murthy Jatavallabhula, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, Liam Paull, Florian Shkurti, Derek Nowrouzezahrai, and Sanja Fidler. gradSim: Differentiable simulation for system identification and visuomotor control. In *International Conference on Learning Representations (ICLR)*, 2021.

Hanxiao Jiang, Hao-Yu Hsu, Kaifeng Zhang, Hsin-Ni Yu, Shenlong Wang, and Yunzhu Li. PhysTwin: Physics-informed reconstruction and simulation of deformable objects from videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7219–7230, October 2025.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. Incremental potential contact: Intersection- and inversion-free, large-deformation dynamics. *ACM Transactions on Graphics*, 39(4):49:1–49:20, 2020. doi: 10.1145/3386569.3392425.

Yifei Li, Tao Du, Kui Wu, Jie Xu, and Wojciech Matusik. DiffCloth: Differentiable cloth simulation with dry frictional contact. *ACM Transactions on Graphics*, 42(1):2:1–2:20, 2023. doi: 10.1145/3527660.

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

Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics*, 32(6):209:1–209:7, 2013.

Miles Macklin. Warp: A high-performance Python framework for GPU simulation and graphics. NVIDIA GPU Technology Conference (GTC); software framework, `https://github.com/NVIDIA/warp`, 2022.

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. XPBD: Position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games (MIG)*, 2016.

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Transactions on Graphics*, 23(3):449–456, 2004.

Luke Metz, C. Daniel Freeman, Samuel S. Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007. doi: 10.1016/j.jvcir.2007.01.005.

Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. Expressive body capture: 3D hands, face, and body from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *International Conference on Machine Learning (ICML)*, 2020.

Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Transactions on Graphics*, 37(2):12:1–12:15, 2018. doi: 10.1145/3180491.

Tuur Stuyck and Hsiao-yu Chen. DiffXPBD: Differentiable position-based simulation of compliant constraint dynamics. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 6(3):51:1–51:14, 2023. doi: 10.1145/3606923.

Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning (ICML)*, pp. 20668–20696, 2022.

Jingwei Tang, Vinicius C. Azevedo, Guillaume Cordonnier, and Barbara Solenthaler. Honey, I shrunk the domain: Frequency-aware force field reduction for efficient fluids optimization. *Computer Graphics Forum*, 40(2), 2021.

Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C. Karen Liu. Fast and feature-complete differentiable physics engine for articulated rigid bodies with contact constraints. In *Robotics: Science and Systems (RSS)*, 2021. doi: 10.15607/RSS.2021.XVII.034.

## A Proof of Theorem 1

Equation (5) is the chain rule applied to $S = S_n \circ \cdots \circ S_1$, so the adjoint visits blocks in reverse order. For a single block, the $C^1$ hypothesis on $F_i$ together with invertibility of $\partial F_i/\partial \Delta \boldsymbol{x}_i$ at the executed linearization state gives Eq. (7) by the implicit function theorem. Since $\boldsymbol{x}_i^+ = \boldsymbol{x}_i + \Delta \boldsymbol{x}_i(\boldsymbol{x})$, the local vector–Jacobian product splits into the direct contribution of Eq. (8) and the implicit contribution; introducing $\boldsymbol{p}_i$ through Eq. (9) rewrites the latter as Eq. (10). Finally, the implementation updates all vertices of one color in parallel; distance-1 coloring makes same-color updates touch disjoint blocks through disjoint stencils, so the parallel update equals their composition in any order and the same holds for the adjoints. Composing in reverse yields the claim. □

## B The construction on other block solvers

Theorem 1 asks only for a finite composition of differentiable implicit updates with invertible linearized systems – block-local or, as in projective dynamics' global step, whole-state, so it should apply beyond Vertex Block Descent. We verify this on minimal instantiations of two further solvers; both implementations ship with the released code.

*Extended position-based dynamics*: the block is one distance-constraint projection (a closed-form update of one Lagrange multiplier and two particles), swept sequentially, $K$ sweeps per step; the reverse sweep replays the projections in reverse with a hand-derived local vector–Jacobian product. *Projective dynamics*: one iteration is a per-spring local projection followed by a global solve with a *constant* prefactored symmetric positive-definite matrix; the backward reuses the same factorization (transpose solve) and hand-derived projection vector–Jacobian products, so the backward mirrors the forward's local–global structure exactly.

Both are validated the same way as the main experiments: against `torch.autograd` through the identical executed forward, on spring and constraint networks built over the §6.1 test meshes ($|V|$=26 and 115), at $K \in \{1, 5, 20\}$ (XPBD: extended position-based dynamics; PD: projective dynamics):

| Solver | $\lvert V\rvert$ | $K$=1 | $K$=5 | $K$=20 |
|---|---|---|---|---|
| XPBD | 26 | $3.9 \times 10^{-16}$ | $4.8 \times 10^{-16}$ | $7.0 \times 10^{-16}$ |
| XPBD | 115 | $6.2 \times 10^{-16}$ | $6.0 \times 10^{-16}$ | $5.0 \times 10^{-16}$ |
| PD | 26 | $9.3 \times 10^{-16}$ | $9.7 \times 10^{-16}$ | $9.7 \times 10^{-16}$ |
| PD | 115 | $2.5 \times 10^{-15}$ | $5.9 \times 10^{-15}$ | $1.1 \times 10^{-14}$ |

Worst case $1.1 \times 10^{-14}$ (the projective-dynamics global solve adds a factorization round-trip per iteration), i.e. the same finite-depth exactness as the Vertex Block Descent instantiation. These are exactness demonstrations, not performance claims: the implementations are deliberately plain NumPy.

## C Closed forms for the local adjoint

Eq. (12) gives the parameter contribution abstractly. For the block Hessian actually used, $H_i^{\text{elas}} = \sum_{e \ni i} w_e \big(\mu \|\nabla N_{i,e}\|^2 I + \lambda\, q_{i,e} q_{i,e}^\top\big)$, with $w_e = \mathrm{vol}_e h^2$ the element weight in the $h^2$-scaled system and $\nabla N_{i,e}$ the constant shape-function gradient, the Hessian-tangent contraction of Eq. (11) collapses to scalars, with no tensor assembled and no derivative of $q$ required:

$$\left(\frac{\partial H_i^{\text{elas}}}{\partial \mu}\right) : \left(\boldsymbol{p}_i \Delta \boldsymbol{x}_i^\top\right) = \sum_{e \ni i} w_e \|\nabla N_{i,e}\|^2 \left(\Delta \boldsymbol{x}_i \cdot \boldsymbol{p}_i\right), \tag{14}$$

$$\left(\frac{\partial H_i^{\text{elas}}}{\partial \lambda}\right) : \left(\boldsymbol{p}_i \Delta \boldsymbol{x}_i^\top\right) = \sum_{e \ni i} w_e \left(q_{i,e}^\top \Delta \boldsymbol{x}_i\right)\left(q_{i,e}^\top \boldsymbol{p}_i\right). \tag{15}$$

Combining these with the parameter derivatives of the stable Neo-Hookean energy as implemented, the reparameterized form $\Psi = \frac{\mu}{2}(\|F\|_F^2 - 3) + \frac{\bar{\lambda}}{2}(J - \alpha)^2$ with $\bar{\lambda} = \mu + \lambda$, $\alpha = 1 + \mu/\bar{\lambda}$ (Smith et al., 2018), whose Piola derivatives are $\partial P/\partial \mu = F + (J-2)\,\mathrm{cof}\, F$ and $\partial P/\partial \lambda = (J-1)\,\mathrm{cof}\, F$ (equivalently $\partial V_e/\partial \mu = \frac{1}{2}(\|F_e\|_F^2 - 3) + \frac{1}{2}(J_e - 2)^2$ and $\partial V_e/\partial \lambda = \frac{1}{2}(J_e - 1)^2$, each up to a

state-independent constant) closes the material gradient: one dot product and one quadratic form per incident element, reusing the $\boldsymbol{p}_i$ already computed for the state adjoint.

Two consequences are worth stating. First, the cost of material gradients is a constant multiple of the state adjoint, which is why enabling them does not change the asymptotic backward cost. Second, Eqs. (14)–(15) are precisely the terms that defect (ii) of §3.5 drops: an implementation that keeps them for parameters but not for the state adjoint produces material gradients that look correct while the state gradient carries error $\propto \|\Delta \boldsymbol{x}\|$.

## D SUPPORTING FIGURES AND TABLES

**Table 5: Differentiation strategies for implicit solvers.** *Executed solver?*: does the backward operate on the same finite block sweep the forward ran, rather than on a converged equation or an unrolled implementation graph. *Global object*: the global linear-algebra structure the backward constructs, if any. *Block-local?*: is the backward a per-block solve in the forward's color partition. *Memory vs. $K$*: asymptotic in solver depth at fixed mesh size; our constant of $\approx$4 kilobytes per vertex per sweep is measured (App. K). *Scale*: largest configuration reported in each work.

| Method | Executed solver? | Global object | Block-local? | Memory vs. $K$ | Scale |
|---|---|---|---|---|---|
| Unrolled AD | graph level | none | no | linear in graph | $\sim$10k DoF |
| Equation-level impl. diff. | no | sparse adj. | no | $O(1)$ | varies |
| DiffPD / DiffCloth | no | PD Cholesky | no | $O(1)$ | $\sim$61k DoF |
| DiffXPBD | per-step eqs. | sparse adj. (CG) | no | linear in states | 66k verts |
| DiffQN | no | quasi-Newton | no | $O(1)$ | $\sim$158k DoF |
| DiffIPC | no | step Hessian | no | $O(1)$ | $\sim$54k verts |
| **Ours** | **yes** | **none** | **yes** | **lin., $\approx$4 KB/v/sw** | **8M verts** |

This appendix collects the qualitative results and supporting tables that do not fit the main-text page budget. ICLR does not count appendices against the page limit, so nothing is omitted for space.

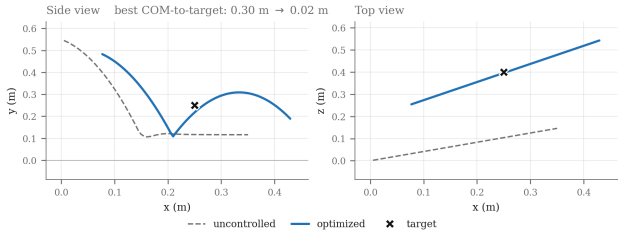


**Figure 3:** Cube center of mass trajectory under the initial parameters versus the optimized ones, side and top view. The optimizer, not the initialization, drives the target hit.

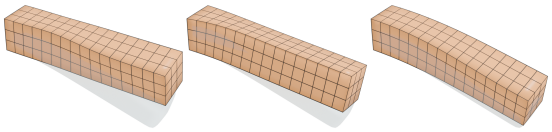

**Figure 4:** Beam $\mu/\lambda$ identification, target deformation $(\mu, \lambda)$=$(800, 400)$ under a $60^\circ$ angular kick. Blue ghost: target equilibrium. Both parameters are recovered to $< 0.3\%$.

## E ADDITIONAL INVERSE PROBLEMS

Three further inverse problems, each a scalar or low-dimensional recovery by backpropagation through the full rollout, are reported here rather than in §6.5 for space. They are included because they probe a regime the main-text tasks do not: the optimized quantity is not a material constant or an initial state but a *rest-shape* parameter (the dress scale below) or a persistent external force (the cape wind), and in both cases the gradient must survive pinning constraints and contact along the whole trajectory. Table 7 summarizes them.

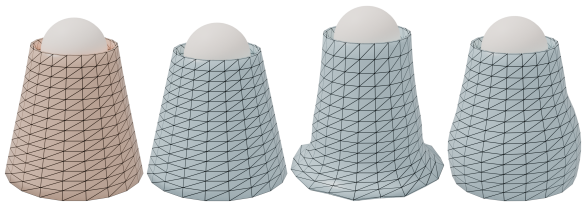

**Figure 5:** Dress $\mu/\lambda$ identification. Left: initial stiff configuration ($\mu$=5000). Remaining panels: target draping progression ($\mu$=1500).

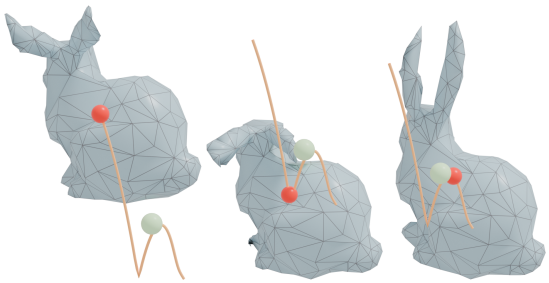

**Figure 6:** Bunny initial-state optimization: start, mid-bounce, and at the target.

**Shear modulus from a scalar sag target.** A 484-vertex square cloth (side 0.6 m, extruded to a thin three-tet shell) is pinned at its four corners and sags into a hammock under gravity. Stiffer $\mu$ gives shallower center sag, so the inverse problem $\mathcal{L}(\mu)=(y_{\text{ctr}}(\mu)-y^{\star}_{\text{ctr}})^2$ is scalar and monotonic. From an over-stiff initialization $\mu_0$=8000 Pa against a target $\mu^{\star}$=2000 Pa, Adam (learning rate 0.10, cosine decay, 80 iterations) on $\partial\mathcal{L}/\partial\mu$, obtained from the per-tetrahedron parameter-gradient outputs of the reverse sweep, summed over tetrahedra, recovers $\hat{\mu}$=1996.7 Pa (0.16% error) and reduces the loss by five orders of magnitude (Fig. 8).

**Rest-shape size from a hem-height target.** A 924-vertex procedural A-line dress is pinned at an SMPL-X (Pavlakos et al., 2019) neckline ($y$=1.45 m) and drapes through 35 DiffVBD steps. A single scalar *scale* $s$ multiplies all rest-pose radii, so the optimized quantity enters through the rest metric rather than through the material or the initial state; the target is the hem height $y^{\star}_{\text{hem}}$=0.654 m under $\mathcal{L}(s)=(y_{\text{hem}}(s)-y^{\star}_{\text{hem}})^2$. Eighty Adam iterations (learning rate 0.025) drive $\mathcal{L}$ from $3.4\times10^{-2}$ to $1.8\times10^{-7}$ and recover $\hat{s}$=0.910 to within 0.5% (Fig. 9).

**Constant wind force from a hem-shape target.** An 864-vertex cape pinned along its top edge is exposed to an unknown constant wind term $\boldsymbol{w}$ added to free-vertex velocities each step (a per-step velocity increment, in m/s). Given the target hem shape produced by $\boldsymbol{w}^{\star}=(2.5,0,1.5)$, we recover $\boldsymbol{w}$ from a zero-wind start with 120 Adam iterations on $\mathcal{L}(\boldsymbol{w})=\|\boldsymbol{x}_T(\boldsymbol{w})-\boldsymbol{x}^{\star}_T\|^2$. Each step's adjoint (§3.4) returns $\partial\mathcal{L}/\partial\boldsymbol{v}_{\text{pre}}$, which chains to the wind by $\boldsymbol{w}\leftarrow\boldsymbol{w}-\eta h\sum_i\bar{\boldsymbol{v}}_i$. Final relative error is below 0.3%, with the loss falling from $4\times10^{-2}$ to $2\times10^{-8}$ (Fig. 10). This task differs from the flag-wind task of §6.5, which recovers a 24-scalar time-varying wind on a different mesh and is under-determined; here the wind is constant and the recovery is unique.

## F Gradient validation against finite differences

Independently of that reference, we validate against central finite differences on a single-tetrahedron scene, sweeping the *relative* FD step $\epsilon\in[10^{-3},10^{-8}]$ for $\boldsymbol{x}_0$, $\boldsymbol{v}_0$, $\mu$ and $\lambda$. The quantity reported is the maximum relative error between the analytic and FD gradients; a configuration passes if that error is below $10^{-6}$ and stable across the $\epsilon$ sweep (so that it reflects the gradient, not FD truncation or round-off). All 11 checks pass; Table 6 (App. D) lists six representative rows. CPU and GPU agree to machine precision.

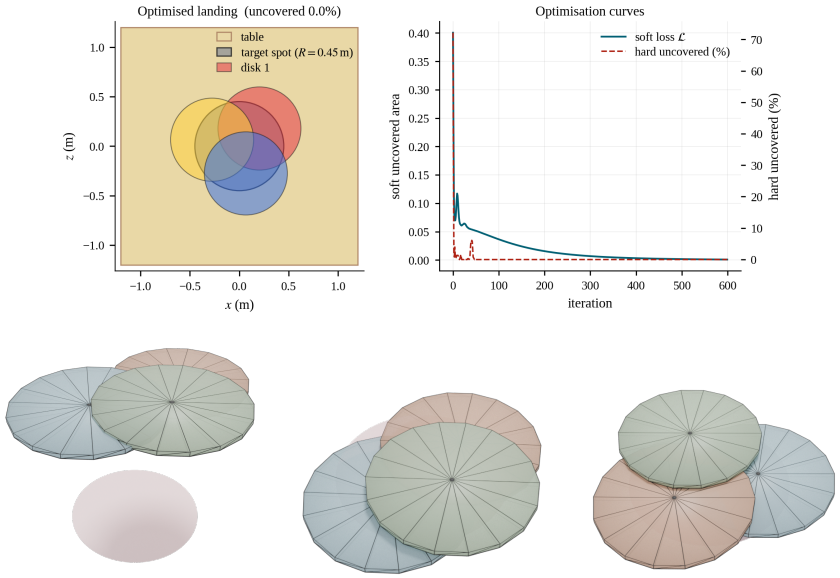


**Figure 7: Cover the Spot.** Three soft-body disks are jointly optimized by backpropagation through the full DiffVBD rollout to land such that their union covers a painted target spot. *Top:* top-down view of the optimized landing (left) and optimization curves (right). *Bottom:* 3D snapshots of the optimized throw at three moments: high mid-air, low mid-air, and settled. Final hard uncovered area: $0.0\%$ (full coverage).

**Table 6:** Finite-difference (FD) validation. Maximum relative error between analytic and FD gradients. The improvement with $K$ reflects the executed solver approaching a stationary point, where the FD reference is best conditioned.

| Configuration | Max relative error | Pass |
|---|---|---|
| Baseline ($K$=1, CPU) | $4.07 \times 10^{-10}$ | ✓ |
| Baseline ($K$=1, GPU) | $4.07 \times 10^{-10}$ | ✓ |
| Multi-sweep $K$=2 | $1.64 \times 10^{-9}$ | ✓ |
| Multi-sweep $K$=3 | $3.05 \times 10^{-10}$ | ✓ |
| $\mu$-only ($\lambda$=0) | $1.34 \times 10^{-9}$ | ✓ |
| $\lambda$-only ($\mu$=0) | $5.33 \times 10^{-10}$ | ✓ |

## G SOLVER DEPTH AND ACHIEVED OUTCOME

Deeper solvers give more accurate gradients, but the operative question is whether that changes what an optimizer achieves. Whether better gradients help the optimizer at all is subtle in general (Suh et al., 2022). Repeating the cube initial-state task at $K \in \{5, 20, 100\}$ under identical 40-iteration Adam budgets from the default initialization gives best center-of-mass-to-target distances of $0.093$, $0.061$ and $0.034$ m (Fig. 11). All three land far closer than the uncontrolled trajectory, and in this run deeper solvers land closer, but a single initialization is uninformative either way: with six optimization variables and a short budget the gap between depths could easily be smaller than the spread induced by where the optimizer starts.

We therefore measured that spread. The optimization is deterministic given an initialization, so we re-ran the full sweep from 8 initializations drawn around the default with $15\%$ relative jitter, holding everything else fixed (Table 8).

Two things follow, and we state both. The mean does improve from $K$=5 to $K$=20 ($0.052 \rightarrow 0.034$ m) and then flattens, with $K$=20 and $K$=100 statistically indistinguishable (Welch $t$=0.11). But the improvement from $K$=5 is only borderline resolved at this sample size (Welch $t$=2.16, df$\approx$10.5 against $K$=100; $t$=1.87 against $K$=20), so we do *not* claim a significant depth effect on the achieved objective. The clearer effect is on variance: the standard deviation falls by $1.9\times$ from $K$=5 to $K$=100, i.e. deeper solves make the outcome more repeatable across initializations even where they do not make it better on average. The claim we do make remains the one in §6.1,

**Table 7:** Additional inverse problems. All use the DiffVBD adjoint with Adam. Loss is the task objective at convergence; error is relative to the known ground truth used to generate the target.

| Task | #vars | Iters | Loss (start → end) | Outcome |
|---|---|---|---|---|
| Cape $\mu$ (center sag) | 1 | 80 | $1.67\times10^{-2} \rightarrow 1.64\times10^{-7}$ | $\hat{\mu}$=1996.7 Pa (0.16%) |
| Dress size (hem height) | 1 | 80 | $3.4\times10^{-2} \rightarrow 1.8\times10^{-7}$ | $\hat{s}$=0.910 (<0.5%) |
| Cape wind (hem shape) | 3 | 120 | $4\times10^{-2} \rightarrow 2\times10^{-8}$ | <0.3% relative |

*(top) Init $\mu_0$=8000 Pa (over-stiff, hammock barely sags); (bottom) recovered $\hat{\mu}$=1996.7 Pa (target $\mu^\star$=2000, err 0.16%)*

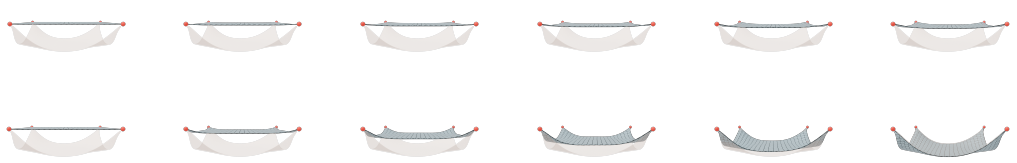

**Figure 8: Shear modulus recovery on a four-corner-pinned cape.** The translucent silhouette below the cloth is the optimization target (deeper sag). Frames at $t \in \{0.02, 0.08, 0.15, 0.22, 0.28, 0.33\}$ s.

about gradient accuracy, which is measured against a reference rather than inferred from optimizer outcomes.

## H Implementation details

**Backends.** Two GPU implementations share one algorithm. The first is written in NVIDIA Warp (Macklin, 2022) kernels; the second is a fused CUDA C++ extension compiled on first use through PyTorch's `load_inline` and cached thereafter. Both are `float64` end to end. They agree to $10^{-16}$ on forward state and on gradients, which is what lets us use either as the reference for the other; the fused path is 7–22× faster because one kernel launch per color performs assembly, the 3×3 solve, the saturation and the write-back without a round trip to global memory between stages.

**Data layout.** Each vertex owns a *star*: the list of (tetrahedron, local vertex index) pairs it participates in, packed into flat offset/element/local-index arrays so a thread reads a contiguous range. The inverse rest metric of each tetrahedron is stored as nine separate scalar arrays $\mathrm{IB}_{00} \ldots \mathrm{IB}_{22}$ rather than as a 3×3 struct, so that a 32-thread hardware warp reading one column is coalesced. Vertices are distance-1 greedily colored, which guarantees that two vertices of one color never share a tetrahedron and can therefore be updated in parallel without write conflicts.

**Replay buffer.** The backward does not store an AD graph. Per sweep $k$ and vertex $i$ it stores only the applied update $\Delta\boldsymbol{x}_i^{(k)}$ (three doubles) and the six independent entries of the symmetric block $H_i^{(k)}$, that is nine `float64` per vertex per sweep, or 72 bytes. Everything else the reverse sweep needs is recomputed from the replayed state. The ≈4 KB per vertex per sweep quoted in Table 9 is the caching-allocator peak (`torch.cuda.max_memory_allocated`), which also contains the mesh, the star index and the material arrays; the tape proper is the 72 bytes. The reverse pass walks sweeps, colors and vertices in exactly reversed order, undoing $\Delta\boldsymbol{x}_i$ to recover the pre-update state rather than storing it.

**Safeguards.** Two safeguards in the forward are differentiated rather than treated as constants, which is what §3.5 requires. The per-vertex block is regularized as $H_i + \tau I$ with $\tau$=$10^{-6}$ in the $h^2$-scaled system the kernels assemble (equivalently $\tau/h^2$ in the normalization of Eq. (2)), and every update is saturated by $\Delta\boldsymbol{x} \leftarrow \Delta\boldsymbol{x}\, c/(\|\Delta\boldsymbol{x}\| + c + \varepsilon)$ with $c$=0.1 m and $\varepsilon$=$10^{-12}$. Both constants are defined once as compile-time macros shared by the forward and backward kernels, so the two cannot drift apart. Two details matter for a faithful re-implementation of Algorithm 1: the saturation Jacobian of Eq. (13) is applied only to the copy of $\bar{\boldsymbol{x}}_i^+$ that enters the local adjoint solve (the identity branch of Eq. (8) uses the untransformed adjoint), and the Hessian-tangent and parameter contractions use the *raw* update $\Delta\boldsymbol{x}_i^{\mathrm{raw}}$, recovered exactly by inverting the saturation of the recorded step. Writing them

*(top) Init dress scale $s_0$=0.45 (too short, hem above waist); (bottom) recovered $\hat{s}$=0.910 (hem at mid-thigh, err <0.5%)*

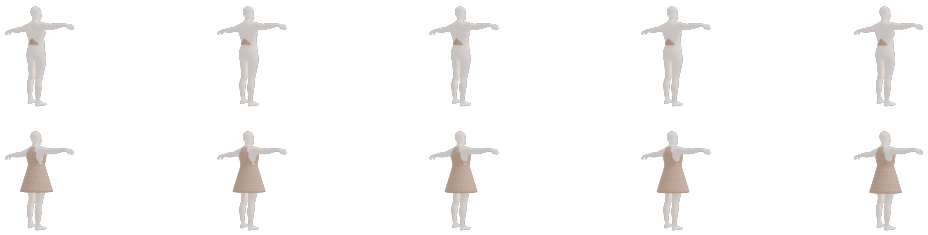

**Figure 9: Fitting a procedural dress to a body by optimizing a rest-shape scale.** Frames at $t \in \{0.02, 0.13, 0.25, 0.37, 0.58\}$ s.

*(top) Init $\boldsymbol{w}_0$=**0**; (bottom) recovered $\hat{\boldsymbol{w}}$=(2.49, 0, 1.49)*

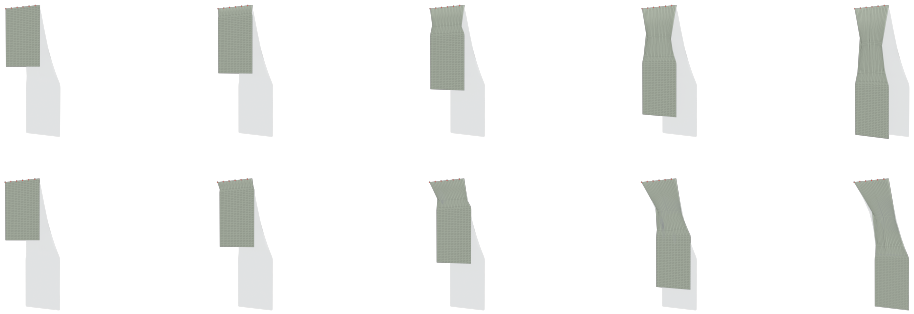

**Figure 10: Wind-force recovery on a top-pinned cape.** Frames at $t \in \{0.02, 0.13, 0.27, 0.40, 0.50\}$ s.

exactly matters: Warp through version 1.11 rounds a literal inside `wp.float64(0.1)` through `float32` first, which perturbs $c$ by $1.5 \times 10^{-8}$ relative and was large enough to move the Warp backend off machine precision until the constants were written exactly.

**Contact.** Two primitives coexist. The built-in self and floor model is an IPC (Li et al., 2020) log barrier with stiffness $\kappa$, activation distance $\hat{d}$ and a lagged dissipation term $c_{\mathrm{damp}}$ that damps only downward motion, so the damping does not fight separation. For many-body scenes a differentiable center-of-mass penalty acts between bodies: a uniform-grid fixed-radius search (cell size equal to the search radius, linearized cell keys sorted once, candidates from the 27 neighbor cells by binary search) supplies candidate pairs, and a ReLU (optionally softplus, sharpness $\alpha$) penalty on the overlap supplies the force, differentiable in the centers of mass and therefore in the vertex positions that define them. The pair list is index data with no gradient, so the broad phase adds nothing to the adjoint.

**Tensor interface and large runs.** Both backends expose forward and backward entry points that consume and return GPU-resident tensors, so an outer optimization loop never copies through the host; this is roughly $4\times$ faster than the array-based interface for backpropagation-through-time loops. At $10^6$ bodies the rollout is gradient-checkpointed (Chen et al., 2016) every 5 steps, bodies are packed into a single mega-mesh processed in chunks of 65,536, and gradients are clipped at norm 5; material parameters are held fixed in the $10^6$-body runs.

## I Inverse-problem tasks in detail

These are the per-task descriptions for §6.5; the headline numbers are Table 4 in the main text, and the reproduction protocols are Appendix J.

**Material identification.** Given a target rollout produced by unknown Lamé parameters, recover them. On a 208-vertex cantilever twisted 180° and released, Adam with backpropagation through time recovers $\mu_{\mathrm{opt}}$=1.001 $\times$ $10^6$ from an initialization $10\times$ too soft (0.1% error, 76 iterations

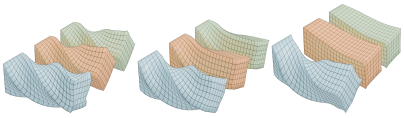

**Figure 12:** Beam $\mu$ identification at three twist stages. Blue: initial ($\mu_{\mathrm{init}}$=$10^5$). Coral: target ($\mu_{\mathrm{true}}$=$10^6$). Green: recovered.

**Table 8: Initialization spread of the cube initial-state task**, best center-of-mass–target distance in metres over 8 perturbed initializations per depth, identical 40-iteration budgets. Deeper solves are better on average and roughly twice as consistent, but the mean gap is at the edge of what 8 initializations resolve.

| $K$ | mean | std | min | max |
|---|---|---|---|---|
| 5 | 0.052 | 0.020 | 0.023 | 0.081 |
| 20 | 0.034 | 0.018 | 0.015 | 0.065 |
| 100 | 0.035 | 0.010 | 0.021 | 0.053 |

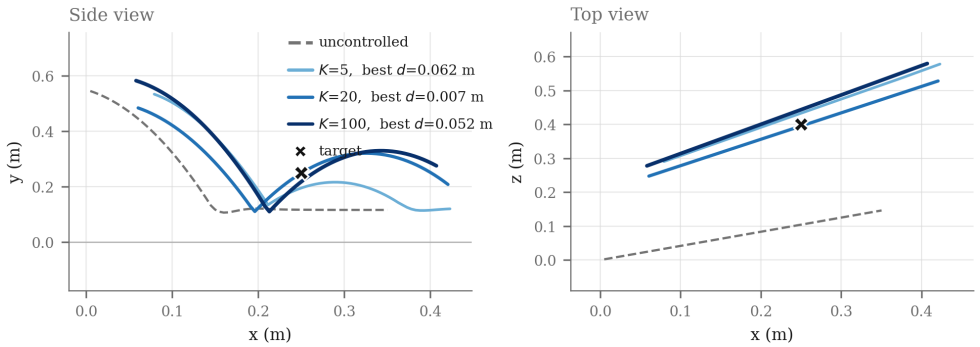


**Figure 11:** Optimized cube center of mass trajectories at three solver depths under identical 40-iteration Adam budgets, with the uncontrolled trajectory under the initial parameters overlaid (gray dashed). Every optimized trajectory passes close to the target; the ordering between depths is within run-to-run spread on this task (App. G).

to the $10^{-4}$ stopping criterion). With both parameters free, $(\mu, \lambda)$=$(800, 400)$ is recovered from $(300, 150)$ to $< 0.25\%$ (Figs. 12, 4). On a 600-vertex dress draping over a capsule mannequin with IPC ground contact, single-stage log-space Adam recovers $(1471, 880)$ from $(5000, 3000)$ (Fig. 5); replacing the vertex-position loss with a radial silhouette profile gives $(1528, 900)$.

The dress pair is recovered by a two-stage schedule whose structure mirrors the solver's own block-coordinate principle. No single observation window identifies both parameters: the time-averaged settled drape pins $\mu$ (to $0.4\%$ under a 1-D Newton) but leaves $\lambda$ nearly free, while per-step matching of the early frames, before the contact-rich trajectory decoheres (Metz et al., 2021) – pins $\lambda$ (to $\approx 1\%$) but constrains $\mu$ weakly from the soft side. Stage A runs Adam on the sum of both objectives, which moves the pair together into the correct basin; stage B then alternates damped 1-D Newton solves, $\mu$ against the settled-shape objective and $\lambda$ against the early-dynamics objective. The alternation alone is insufficient: started from $(5000, 3000)$ it locks into a self-consistent wrong fixed point $(1325, 3310)$, which is why the basin-entering stage comes first. From the same initialization the full schedule reaches $(1499.9, 800.2)$ against the true $(1500, 800)$, bit-identical over three repeated runs. The same schedule transfers to the silhouette variant with every observation restricted to radial profiles (a silhouette *sequence* rather than a single settled silhouette): it recovers $(1500.0, 800.0)$ exactly, and although the stage-A iterates differ slightly across repeated runs, stage B contracts all of them to the same fixed point, so the answer is run-to-run identical.

**Initial-state optimization.** Given a target voxel, optimize $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ so the center of mass reaches it after ground contact. The best center-of-mass–target distance over the rollout drops from $0.304\,\mathrm{m}$ under the initial parameters to $0.013\,\mathrm{m}$ for a cube (8 vertices, 6 tetrahedra), confirming that the optimization, not the initialization, drives the result. The bunny starts much closer to its target ($0.035\,\mathrm{m}$) and ends at $0.034\,\mathrm{m}$, so that task exercises the gradient without demonstrating a large improvement (Figs. 13, 6; Fig. 11 overlays the uncontrolled baseline).

**Wind control.** Given a target flag trajectory, recover a spatially-uniform, time-varying wind on a 442-vertex flag pinned along its top edge, parameterized as piecewise-constant over 8 equal time windows ($8 \times 3 = 24$ scalars, dimensionless force units relative to a unit reference wind). Loss falls from $3.2 \times 10^{1}$ to $5.1 \times 10^{-3}\,\mathrm{m}^2$ in 199 iterations (Fig. 14). The recovered wind differs from the reference by $\|\Delta w\| = 15.3$ because the problem is under-determined: 24 wind scalars act through

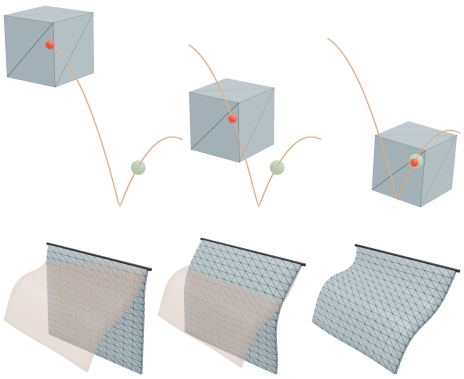

**Figure 13:** Cube initial-state optimization: start, mid-bounce, and near target. The center of mass trajectory comparison is Fig. 3.



**Figure 14:** Flag under the recovered time-varying wind, at three times in the rollout.

$\sim$12 effective trajectory degrees of freedom, so trajectory matching does not identify the wind uniquely.

**Contact-rich planning (Cover the Spot).** Three soft disks ($r$=0.42, $r/R$=0.933, just above the $R\sqrt{3}/2$ infeasibility bound) drop from 2.5 m onto a target of radius $R$=0.45 m. All three share one mega-mesh in a single Vertex Block Descent context, so disk–disk and disk–floor interactions are resolved by the IPC log barrier ($\kappa$=400, $\hat{d}$=2 cm, lagged damping $c_d$=8; frictionless). We jointly optimize per-disk $\boldsymbol{v}_0$ and a small $\boldsymbol{x}_0$ offset (18 scalars) under a smooth coverage objective with sigmoid sharpness annealed $4 \rightarrow 40$; each forward rolls 160 steps with full backpropagation through time, so adjoints flow through every barrier evaluation. 600 Adam iterations in about two minutes reduce the hard uncovered fraction from 72% to $\mathbf{0.0}$% (App. D, Fig. 7).

## J TASK PROTOCOLS

Every task below is a script in the released harness and is run by a single command; scenes are constructed procedurally, so there are no data dependencies. We state what is given, what is optimized, and what counts as success. Reported values are read out of the run logs by the harvesting script rather than transcribed by hand.

**Beam $\mu$ identification.** *Given*: a target rollout of a $13 \times 4 \times 4$ tetrahedralized cantilever ($1.0 \times 0.2 \times 0.2$ m), twisted 180° and released, simulated at $h$=2 ms for 100 steps with $K$=15 under a known $\lambda$=$10^6$. *Optimized*: the scalar $\mu$, from an initialization 10× too soft ($10^5$ against a true $10^6$), by Adam on $\partial\mathcal{L}/\partial\mu$ with the per-tetrahedron parameter gradients summed. *Criterion*: relative error in the recovered $\mu$.

**Beam $\mu, \lambda$ identification.** As above but with both Lamé parameters free and a 60° angular kick as excitation; initialization $(300, 150)$ against a true $(800, 400)$. *Criterion*: relative error in both parameters.

**Dress $\mu, \lambda$ and dress silhouette.** *Given*: a target drape of a garment mesh over a capsule mannequin with IPC ground contact. *Optimized*: $(\mu, \lambda)$ in log space, so that the iterates stay positive without clipping. The two variants differ only in the loss: a per-vertex squared position error, or a radial silhouette profile that discards the correspondence and compares only the outline. *Criterion*: relative error against the parameters that generated the target.

**Flag wind control.** *Given*: a target trajectory of a 442-vertex flag pinned along its top edge, rolled out for 120 steps at $K$=10. *Optimized*: a spatially uniform wind held piecewise constant over 8 equal time windows, i.e. 24 scalars, by Adam at learning rate 2.0. *Criterion*: squared trajectory error. We additionally report $\|\Delta w\|$ against the generating wind, which stays large because the map from 24 wind scalars to the trajectory is not injective; the task is trajectory matching, not wind identification, and we label it as such.

**Cube and bunny initial state.** *Given*: a target voxel. *Optimized*: the initial position offset and velocity (6 scalars for the cube, 3 for the bunny) by Adam with gradient clipping, over a rollout that

includes floor contact. *Criterion*: the closest center-of-mass approach to the target over the rollout, compared against the same quantity under the initial parameters, so that the initialization cannot be credited with the result.

**Cover the Spot.** *Given*: a circular target of radius $R$ and three soft disks released from a fixed height. *Optimized*: per-disk initial velocity and a small initial position offset (18 scalars) under a smooth coverage objective whose sigmoid sharpness is annealed during optimization, with all three disks in one mega-mesh so that disk–disk and disk–floor contact is resolved by the IPC barrier and differentiated through every step. *Criterion*: the hard (non-smoothed) uncovered fraction of the target. The task is feasible only when the disk radius exceeds $R\sqrt{3}/2$; below that bound no three disks can cover the target and the script refuses to run, so the radius is part of the protocol and not a free knob.

## K FULL $K \times |V|$ SCALING GRID

The backward-to-forward ratio is not a single number, and its dependence is easy to misread, so we state it explicitly: at 339 vertices the ratio is $4.9\times$ for the Warp backend, dominated by fixed per-color kernel-launch overhead rather than by adjoint arithmetic; the overhead amortizes as $|V|$ grows, and Table 9 reports the full grid.

On the bunny mesh (339 vertices, 897 tetrahedra, 40 steps) the Warp backend runs 31.4/154.3 ms per forward/backward step, with the fused CUDA C++ path 7–22$\times$ faster (§5); $\sim$360$\times$ more vertices give $\sim$360$\times$ memory at fixed $K$.

Controlled scaling study on a held-out soft jelly-cube tetrahedral mesh (not used elsewhere), on an NVIDIA RTX A6000. 1-to-8 tetrahedral subdivision gives four resolution levels L0–L3; we roll out 50 implicit steps and backpropagate through all of them and sweep both solver depth $K$ and mesh resolution. Every row is measured.

**Table 9: Measured $K \times |V|$ grid** (RTX A6000, all rows measured). Across the grid the backward/forward time ratio stays in 0.96–1.14 (excluding the $K$=5 first-call warm-up, which includes kernel compilation), and peak memory divided by $K|V|$ stays in 3.6–4.6 kilobytes per vertex per sweep, so linearity in $K|V|$ holds to $\pm$13%. At fixed $K$, $\sim$360$\times$ more vertices costs $\sim$360$\times$ memory.

| Mesh | $K$ | $\lvert V\rvert$ | forward (ms) | backward (ms) | peak (MB) |
|---|---|---|---|---|---|
| L0 | 5 | 558 | 1.4 | 4.3 | 12 |
| L0 | 10 | 558 | 1.5 | 1.6 | 22 |
| L0 | 20 | 558 | 2.7 | 2.7 | 42 |
| L0 | 50 | 558 | 6.4 | 6.2 | 100 |
| L0 | 100 | 558 | 12.5 | 12.2 | 201 |
| L0 | 200 | 558 | 24.8 | 24.0 | 393 |
| L1 | 10 | 3,766 | 2.8 | 3.2 | 149 |
| L1 | 50 | 3,766 | 12.6 | 14.4 | 677 |
| L1 | 100 | 3,766 | 24.9 | 28.3 | 1,336 |
| L2 | 10 | 27,169 | 17.0 | 17.1 | 1,074 |
| L2 | 20 | 27,169 | 31.7 | 32.9 | 2,025 |
| L2 | 50 | 27,169 | 76.0 | 80.5 | 4,926 |
| L3 | 10 | 201,341 | 113.3 | 115.8 | 8,136 |
| L3 | 20 | 201,341 | 214.0 | 224.5 | 15,001 |

This grid measures our method alone; the controlled cross-method comparison is Table 3 (one forward, one GPU, one mesh for all four routes).

## L REAL-TO-SIM PROTOCOL AND ANALYSIS

**Data and mesh.** Each PhysTwin capture supplies per-frame tracked 3D points with correspondence, visibility and motion-validity masks, grasp ("controller") trajectories, and a frame-0 shape prior of surface and interior fill points. We place the tracked points first in the vertex list, append the

fill points, tetrahedralize by Delaunay pruned to an alpha-complex-like subset (tetrahedra whose longest edge exceeds $3\times$ the median point spacing are dropped), and keep the largest connected component: the sloth yields $5{,}673$ vertices and $14{,}847$ tetrahedra, the cloth a thin two-layer slab. Because tracked points are mesh vertices, the loss needs no interpolation. Captures are 30 FPS; the solver takes two implicit steps per frame ($h$=16.7 ms, $K$=8) with a drag factor $e^{-3h}$ matching the dataset's convention.

**Boundary drive and loss.** Vertices within $4$ cm of a controller point at frame 0 are pinned and follow their nearest controller point's recorded translation, linearly interpolated per substep; the solver treats pins as Dirichlet conditions in both the forward and the adjoint. The loss is the visibility-masked mean squared error between simulated and observed tracked points at frame boundaries over the train split; Adam runs on $(\log \mu, \log \lambda)$. One iteration (forward, reverse sweep, update) takes $1.9$ s for the sloth's 268-step train horizon on an RTX A6000.

**Table 10: Real-to-sim identification** on PhysTwin captures (Jiang et al., 2025): mean tracked-point error (cm) vs. the real video, before and after fitting $(\mu, \lambda)$, and with the train-selected model corrections of the "Model corrections" paragraph; held-out split per capture.

| | initial | | fitted | | + corrections | |
|---|---|---|---|---|---|---|
| Capture | train | held-out | train | held-out | train | held-out |
| Stuffed sloth (two-hand stretch) | 6.56 | 13.12 | 1.93 | 2.39 | 1.01 | 1.44 |
| Stuffed zebra (two-hand stretch) | 5.79 | 12.44 | 3.82 | 7.69 | 1.09 | 1.48 |

**Identifiability.** Sweeping $\mu$=$\lambda$ over $\{10^0, \ldots, 10^6\}$ shows a one-sided valley in both captures: too soft is penalized sharply (the sloth sags to 6.6 cm at $10^2$), while above $\sim 10^3$ the error is flat at the tracking noise floor. The grasp imposes deformation kinematically, so the capture bounds compliance from below; distinguishing stiffer materials would need force sensing or free-dynamics excitation. We therefore report recovery into the identified basin rather than a point estimate. Capacity is not the bottleneck either: lifting $(\mu, \lambda)$ to per-tetrahedron fields (the adjoint is per-tet already) and fitting the drag coefficient jointly changes held-out track error by under $2\%$: the residual is capture noise and boundary approximation, which is why two global parameters saturate these captures.

**Model corrections.** Decomposing the fitted sloth's error spatially shows it concentrated near the grasps (vertex error $4.7$ cm in the quartile closest to a pin vs. $0.7$ cm in the farthest), and the observed cloud's lowest point stays on a fixed plane throughout while the simulation penetrates it: the capture includes a table the nominal model lacks. Three corrections, each selected on the train split only, follow: (i) a barrier ground plane just under the frame-0 cloud; (ii) plush-scale density ($25$ kg/m$^3$; the nominal $200$ overweights a stuffed toy by an order of magnitude, and train error is monotone in that correction); (iii) driving each grasp's pinned vertices with the rigid transform of its controller cluster, estimated per frame by Kabsch alignment, instead of a single point's translation; (iv) a thin barrier activation distance ($\hat{d}$=2 mm; the 1 cm default floats a light resting body visibly above the table, and this correction alone halves the cloth's train error). At the identified stiffness these bring the sloth to $1.51$ cm train / $1.43$ cm held-out ($40\%$ below the fitted nominal model) and the cloth's train error to $0.51$ cm; the sloth's headline numbers in Table 10 additionally use the grasp treatment of the next paragraph; here and below, every tracked point is scored (points pruned from the mesh rigidly follow their nearest kept vertex), so the metric matches the dataset's protocol exactly. The stiffness transfers across the density correction because the identified basin is the flat side of the one-sided valley above.

**Solver depth is part of the identified model.** Two further train-selected changes, pinning the $k$=50 mesh vertices nearest each hand cluster (robust to controller keypoints that sit off the object surface) and converging the solver harder (four substeps, $K$=16; colored Gauss–Seidel propagates a boundary update roughly one vertex ring per iteration, so the defaults under-resolve long limbs), fit the train split to $1.22$ cm (sloth) and $0.52$ cm (cloth). A third closes most of the remaining train gap: the hand-keypoint Kabsch frame does not reflect the grip direction, so we fit one constant per-hand orientation correction analytically (the average over train frames of the Procrustes residual between the driven and observed pinned vertices about the hand centroid, ramped in with hand displacement; test frames still see controllers only). Selected on a validation window (the last $20\%$ of train frames), the full configuration reaches $1.01$ cm train / $1.44$ cm held-out on the sloth ($1.09$/$1.48$ cm on a zebra

of the same capture family, whose nominal fit only reaches 3.82/7.69 cm), matching the dataset authors' released per-scene model on the train split ($\approx 1.05$ cm under the same metric). Without the orientation correction, held-out error prefers the under-converged configuration the parameters were identified with: the executed solver's truncation acts as identified damping, so $(\mu, \lambda)$ and the solver schedule transfer together or not at all, a real-data instance of this paper's thesis that the object of identification is the executed solver, not an idealized continuum limit. One structural defect is identifiable from data alone: Delaunay meshing cannot distinguish contact from connection, so limbs resting against the body at frame 0 are bridged by phantom tetrahedra (edges between tracked vertices whose observed gap grows by $>3$ cm for $\geq 8$ train frames). Softening those tetrahedra (deleting them orphans their vertices and scores worse) improves the converged configuration's held-out error by a further $6\%$. Fig. 15 overlays the fitted simulations on the captures.

**Exact sensitivities of chaotic trajectories.** On the most dynamic capture we tried (a lifted rope), the full-horizon adjoint at $\mu \leq 10^1$ faithfully reproduces the genuinely exploding sensitivity of a chaotic trajectory, a direct consequence of the reverse sweep's exactness. Truncated backpropagation (windows of 5–10 frames, forward chain unchanged) then yields the usable descent signal, and the damped grasps of the captures in Table 10 keep their full-horizon adjoints finite, so full-horizon fitting is what the table reports.

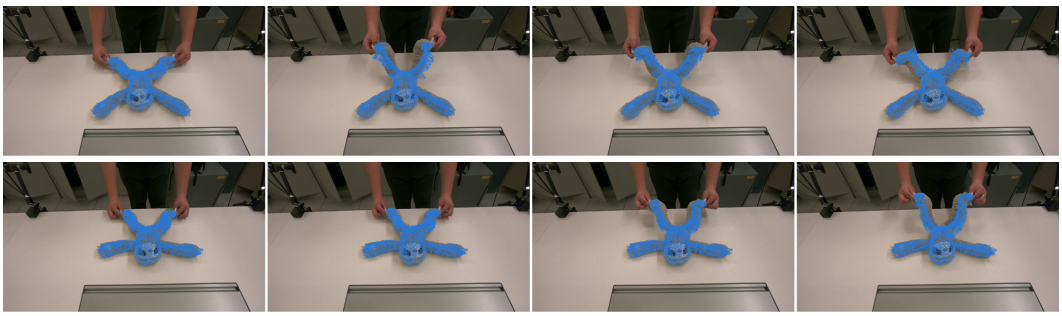

**Figure 15:** Fitted simulations (blue points) projected onto the capture videos at four keyframes: the stuffed sloth under a two-hand stretch (top; keyframes span the full capture; errors in Table 10) and a two-hand lift (bottom; train-window keyframes, train error 1.18 cm), both with the full corrected configuration of App. L.

## M Reproducibility Details

**Hardware and software.** The controlled backward comparison (§6.1–§6.3), the finite-difference checks and the inverse problems were run on a single NVIDIA RTX 4090 Laptop GPU (16 GB) with CUDA 12.1 and PyTorch 2.3, inside one process so that all four backward routes see the same device state. The $K \times |V|$ grid of Table 9 and the real-to-sim fits were run on an RTX A6000, and the $10^6$-body run on one H100 (80 GB). All solver arithmetic is `float64`; no result in this paper depends on mixed-precision behavior.

**Determinism and seeds.** Scene construction is deterministic: meshes, colorings and rest metrics are generated procedurally from fixed integer parameters, so a given task builds the identical problem on every machine. The one deliberate exception is that the backward scatters neighbor contributions with atomic adds, whose summation order is not fixed; run-to-run gradient differences from this are at the level of `float64` round-off and are far below every tolerance reported here, but they mean bitwise reproducibility is not claimed. Random initializations are drawn from seeded generators recorded in the run log.

**What to run.** The released harness maps each numbered claim to a single command; one driver re-runs every experiment and writes one log per experiment, and a harvesting script reads the quoted numbers out of those logs rather than transcribing them. Everything except the $10^6$-body run (§6.7) reproduces in minutes on one consumer GPU.