DIFFGIPC: DIFFERENTIABLE MIXED-COORDINATE CONTACT DYNAMICS ON GPU

Anonymous authors
Paper under double-blind review

ABSTRACT

Contact-rich design and control in robotics require efficient gradients through interactions between deformable and nearly rigid bodies. Representing all components with full-space finite elements (FEM) introduces unnecessary motion degrees of freedom for nearly rigid bodies, while high stiffness makes repeated forward and backward solves costly. We present DiffGIPC, a GPU-accelerated framework for differentiable simulation and gradient-based optimization that couples affine body dynamics (ABD) with FEM-based deformable bodies under Incremental Potential Contact (IPC). We derive gradients for affine rest shapes, initial states, and joint controls while retaining only twelve motion DoFs per affine body in each backward system. The formulation differentiates shape-dependent mass and surface-point maps and retains collision, friction, and joint coupling, allowing one body's parameters to be optimized for their effects on other objects. GPU-parallel derivative evaluation, device-controlled adjoint solves, and local gradient accumulation accelerate the backward pass. A reusable PyTorch interface connects supported physical parameters to user-defined parameter maps and losses across design, identification, motion, placement, and articulated control. Experiments demonstrate peak speedups over DiffIPC of approximately $150\times$ with full-space FEM and over $2900\times$ with affine bodies for combined simulation and gradient evaluation.

1 INTRODUCTION

Gradient-based design, identification, and control require simulators that compute how physical parameters affect motion. From Real2Sim identification to robot motion planning, these tasks often involve contact between deformable and nearly rigid bodies. Parameters and outputs may belong to different objects: a support's shape affects a garment's deformation, while one body's initial velocity changes another body's motion after collision. Computing these dependencies requires differentiating the interactions between objects.

Existing methods face trade-offs in contact modeling and computational cost. ADD (Geilinger et al., 2020) provides analytical sensitivities for multibody dynamics, while DiffIPC (Huang et al., 2024c) differentiates FEM simulation with IPC collision and friction. However, representing nearly rigid components with stiff FEM introduces unnecessary motion DoFs and costly linear solves, and DiffIPC's full-space CPU implementation makes repeated optimization expensive. Differentiable projective dynamics reduces cost through matrix reuse, but contact models can restrict motion: DiffPD permits penetration with penalty contact or enforces sticking in its non-penetration formulation (Du et al., 2021). Efficient differentiation of coupled soft–rigid contact dynamics on the GPU therefore remains an important challenge.

We present DiffGIPC, a GPU-accelerated differentiable framework coupling ABD (Lan et al., 2022) with FEM under IPC (Li et al., 2020). We derive gradients for affine rest shapes, initial states, and joint controls while retaining twelve motion DoFs per affine body in each backward system. The formulation differentiates shape-dependent mass and surface geometry and preserves collision, friction, and joint coupling, allowing one object's parameters to be optimized for their effects on others. Compact coordinates reduce system size, while GPU parallelism accelerates derivative evaluation, assembly, adjoint solves, and gradient accumulation. A reusable PyTorch interface connects sup-

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.

ported physical parameters to user-defined parameter maps and losses without task-specific backward code.

Contributions. Our contributions are (1) a coupled FEM–affine differentiation formulation with affine rest-shape and initial-state derivatives, together with joint differentiation of articulated mechanisms; (2) GPU-parallel backward computation with device-controlled iterative solution; and (3) a reusable optimization interface, demonstrated on a broad set of inverse problems with different parameter and loss choices. Figure 1 summarizes the framework.

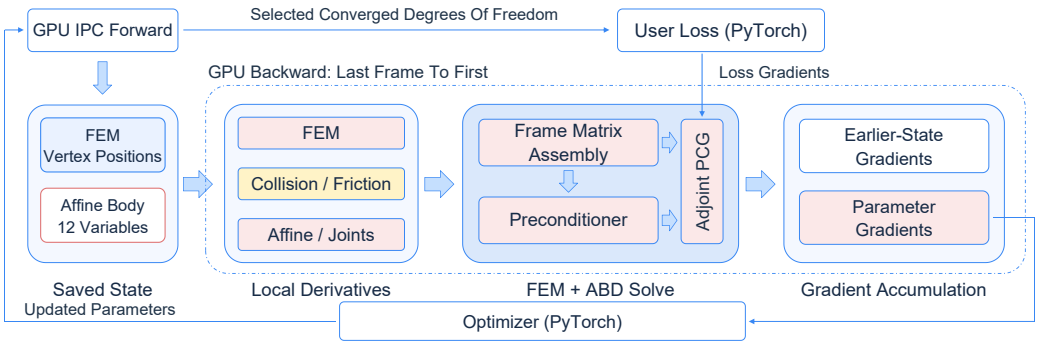


Figure 1: DiffGIPC connects a user-defined loss to the coupled GPU backward calculation. At each frame in reverse order, we assemble one FEM–affine system, compute its adjoint, and accumulate parameter gradients and earlier-frame right-hand-side updates. The derivative of the parameter map converts physical-parameter gradients to optimizer-variable gradients.

2 Related Work

Differentiable mechanics and contact. Differentiable simulation supports identification (Hahn et al., 2019; Jatavallabhula et al., 2021), deformable-body and cloth optimization (Rojas et al., 2021; Liang et al., 2019; Qiao et al., 2020), and rigid-body control (Belbute-Peres et al., 2018; Degrave et al., 2019; Werling et al., 2021). ChainQueen, DiffTaichi, and Brax connect simulation to learning and control (Hu et al., 2019; 2020; Freeman et al., 2021). Applications to assemblies, microstructures, actuators, and garments (Tozoni et al., 2021; Huang et al., 2024b; Gjoka et al., 2024; Li et al., 2025) use different parameters and objectives. Our interface likewise separates implemented physical derivatives from user-defined parameter maps and losses.

To reduce repeated solution costs, projective dynamics (PD) and its dry-friction extensions (Bouaziz et al., 2014; Ly et al., 2020) provide the local–global structure used by DiffPD and DiffCloth (Du et al., 2021; Li et al., 2022). Quasi-Newton methods reduce nonlinear solution cost (Liu et al., 2017); DiffQN reuses forward factorizations with subspace optimization (Cai et al., 2025), and DiffPhD shares a sparse factor across forward, backward, and contact computations for heterogeneous hyperelasticity (Lai et al., 2026). For geometric collision handling, IPC uses a non-penetration barrier (Li et al., 2020), and DiffIPC derives FEM adjoints with collision and friction (Huang et al., 2024c). DiffGIPC retains this equation-level approach but assembles and solves state-dependent, coupled FEM–affine adjoints on the GPU. Contact-energy redesign (Huang et al., 2025b), long-range interactions (Ye et al., 2026), and randomized smoothing (Suh et al., 2022) instead change the contact model or search direction; our focus is the backward computation for the stated IPC model.

Reduced coordinates and GPU computation. Reduction lowers the state dimension using modal bases (Barbič & James, 2005) or adaptive subspaces (Trusty et al., 2024); hyper-reduced PD also approximates constraint projections (Brandt et al., 2018), while subspace mixed FEM addresses material heterogeneity (Trusty et al., 2023). Rigid IPC and ABD use compact coordinates for rigid or nearly rigid motion (Ferguson et al., 2021; Lan et al., 2022), while ADD and SDRS differentiate multibody dynamics and robot shape, respectively (Geilinger et al., 2020; Ye et al., 2025). Beyond reducing motion DoFs, affine shape optimization requires derivatives of rest-dependent mass and

point maps. DiffGIPC includes these derivatives in a coupled adjoint with full-space FEM and twelve motion DoFs per affine body.

GPU execution and preconditioning reduce the remaining computational cost. GIPC and StiffGIPC accelerate IPC (Huang et al., 2024a; 2025a), whereas Fast But Accurate uses local–global computation with complementarity-based frictional contact (Zeng et al., 2025). Multiscale Cholesky and multilevel additive Schwarz preconditioners accelerate the solution of sparse linear systems (Chen et al., 2021; Wu et al., 2022). We inherit StiffGIPC's forward solver and global Hessian assembly. Our backward pass additionally evaluates parameter and history derivatives and solves the coupled GPU adjoints.

3 Mixed-coordinate implicit dynamics

States and design parameters. A free FEM vertex is represented by its position $\mathbf{x}_{t,i} \in \mathbb{R}^3$. Affine body b has a translation $\mathbf{p}_{t,b}$ and an affine transformation $A_{t,b} \in \mathbb{R}^{3\times3}$. The translation and three rows of $A_{t,b}$ form $\mathbf{q}_{t,b} \in \mathbb{R}^{12}$. A surface vertex i has rest position $\bar{\mathbf{x}}_i$ and world-space position

$$\mathbf{x}_{t,i} = \mathbf{p}_{t,b} + A_{t,b}\bar{\mathbf{x}}_i = J_i\mathbf{q}_{t,b}. \tag{1}$$

Here $J_i = \partial\mathbf{x}_{t,i}/\partial\mathbf{q}_{t,b}$ is a 3×12 rest-dependent map. The frame vector $\mathbf{y}_t$ collects the DoFs of N_A free affine bodies and N_F free FEM vertices, giving $n = 12N_\mathrm{A} + 3N_\mathrm{F}$ simulation DoFs. Physical parameters $\boldsymbol{\theta}$—selected rest positions, material and friction coefficients, initial states, and joint controls—are not additional entries of $\mathbf{y}_t$: optimizing more affine rest positions does not increase the twelve motion DoFs in $\mathbf{q}_{t,b}$.

Frame equations. At each step, the converged DoFs satisfy the stationarity equation of the forward incremental potential,

$$\Phi_t = \tfrac{1}{2}\|\mathbf{y}_t - \widehat{\mathbf{y}}_t\|_M^2 + \sum_{e\in\mathcal{E}} w_e(h,\boldsymbol{\theta})\Psi_{t,e}(\mathbf{y}_t, \mathbf{y}_{t-1}; \boldsymbol{\theta}), \qquad \mathbf{g}_t := \nabla_{\mathbf{y}_t}\Phi_t = \mathbf{0}. \tag{2}$$

M contains FEM vertex masses and a 12×12 mass block per affine body. The sum contains elastic, affine-shape, collision, friction, and joint energies with the same weights as the forward simulation. Position-based backward Euler (BDF1) uses $\widehat{\mathbf{y}}_1 = \mathbf{y}_0 + h\dot{\mathbf{y}}_0 + h^2\mathbf{a}_1$ at the first step and $\widehat{\mathbf{y}}_t = 2\mathbf{y}_{t-1} - \mathbf{y}_{t-2} + h^2\mathbf{a}_t$ for $t \geq 2$, where $\mathbf{a}_t$ is the prescribed acceleration. Initial states are simulation inputs, not solved frames. Velocity-reset and static-equilibrium variants are specified in Appendix A.3.

Coupling in world space. For contact c, stack its world-space point positions in $\mathbf{x}_t^c$ and define $K_{t,c} = \partial\mathbf{x}_t^c/\partial\mathbf{y}_t$. The rows for a free FEM vertex contain an identity block; those for a vertex on a free affine body contain J_i in that body's columns. Prescribed points have zero derivatives with respect to the free DoFs. With rest geometry fixed, the collision and current-frame friction gradients and Hessians satisfy

$$\mathbf{g}_{t,c}^y = K_{t,c}^\mathsf{T}\mathbf{g}_{t,c}^x, \qquad H_{t,c}^y = K_{t,c}^\mathsf{T}H_{t,c}^x K_{t,c}. \tag{3}$$

The superscripts x and y denote differentiation with respect to world-space point positions and simulation DoFs, respectively. The maps of all vertices on an affine body use the same twelve body columns. Assembly retains the off-diagonal blocks that couple FEM and affine DoFs, as well as blocks between affine bodies. The collision barrier is $b_\mathrm{bar}(d) = \kappa(d-\widehat{d})^2\log^2(d/\widehat{d})$ for $0 < d < \widehat{d}$ and zero otherwise, where d is the squared distance and $\widehat{d}$ the squared activation distance. Friction uses the normal force $f_n = -2\sqrt{d}\,b'_\mathrm{bar}(d)$ evaluated at the previous configuration. Continuous collision detection limits each forward step to prevent intersections; line search then enforces energy decrease. Appendix A.5 gives the local derivatives and friction-smoothing laws.

4 Coupled differentiation

4.1 Frame-wise adjoint

Write the frame equation as $\mathbf{g}_t(\mathbf{y}_t, \{\mathbf{y}_s : s \in \mathcal{D}(t)\}; \boldsymbol{\theta}) = \mathbf{0}$, where $\mathcal{D}(t) \subseteq \{1, \ldots, t-1\}$ indexes the earlier frames whose states directly enter this equation. At the converged states, define $H_t =$

$\partial \mathbf{g}_t/\partial \mathbf{y}_t$, $C_{t,s} = \partial \mathbf{g}_t/\partial \mathbf{y}_s$, and $G_t = \partial \mathbf{g}_t/\partial \boldsymbol{\theta}$. We evaluate G_t by varying $\boldsymbol{\theta}$ while keeping $\mathbf{y}_1, \ldots, \mathbf{y}_T$ at their converged values. Selected initial states are included in $\boldsymbol{\theta}$, not treated as solved frames. Let $S_t = \mathrm{d}\mathbf{y}_t/\mathrm{d}\boldsymbol{\theta}$. Differentiating the frame equation gives $H_t S_t + \sum_{s \in \mathcal{D}(t)} C_{t,s} S_s = -G_t$. Stacking these equations in frame order gives $\mathcal{J}\mathcal{S} = -\mathcal{G}$, whose explicit block form is given in Equation (43) in Appendix B.1. Each frame equation depends only on the current and earlier states, so $\mathcal{J}$ is block lower triangular, with the coupled frame Hessians H_t on the diagonal and the history derivatives $C_{t,s}$ below it.

For $L(\mathbf{y}_1, \ldots, \mathbf{y}_T; \boldsymbol{\theta})$, let $\boldsymbol{\ell}_t = \nabla_{\mathbf{y}_t} L$ be its partial state gradient and $\boldsymbol{\ell}_\theta$ its partial parameter gradient with all simulated states fixed. The chain rule gives $\nabla_{\boldsymbol{\theta}} L = \boldsymbol{\ell}_\theta + \sum_t S_t^\mathsf{T} \boldsymbol{\ell}_t$. Stack the $\boldsymbol{\ell}_t$ into $\boldsymbol{\ell}_{\mathrm{all}}$ and the frame adjoints into $\boldsymbol{\Lambda}$. Solving $\mathcal{J}^\mathsf{T}\boldsymbol{\Lambda} = \boldsymbol{\ell}_{\mathrm{all}}$ replaces the sum by $\mathcal{S}^\mathsf{T}\mathcal{J}^\mathsf{T}\boldsymbol{\Lambda} = -\mathcal{G}^\mathsf{T}\boldsymbol{\Lambda}$.

Since $\mathcal{J}^\mathsf{T}$ is block upper triangular, backward substitution yields

$$H_t^\mathsf{T} \boldsymbol{\lambda}_t = \boldsymbol{\ell}_t - \sum_{\substack{\tau > t \\ t \in \mathcal{D}(\tau)}} C_{\tau,t}^\mathsf{T} \boldsymbol{\lambda}_\tau, \quad t = T, \ldots, 1; \qquad \nabla_{\boldsymbol{\theta}} L = \boldsymbol{\ell}_\theta - \sum_{t=1}^{T} G_t^\mathsf{T} \boldsymbol{\lambda}_t. \tag{4}$$

We compute the loss gradient without forming $S_t = \mathrm{d}\mathbf{y}_t/\mathrm{d}\boldsymbol{\theta}$. Starting with $\mathbf{b}_t = \boldsymbol{\ell}_t$ and $\bar{\boldsymbol{\theta}} = \boldsymbol{\ell}_\theta$, the reverse pass solves each frame system and accumulates earlier-frame right-hand sides and parameter gradients (Algorithm 1). Local derivative products give these updates without assembling full $C_{t,s}$ or G_t matrices. Only one frame Hessian is needed at a time.

Inertia and friction history. For earlier solved frames, BDF1 gives $C^{\mathrm{in}}_{t,t-1} = -2M$ for $t \geq 2$ and $C^{\mathrm{in}}_{t,t-2} = M$ for $t \geq 3$. Friction additionally depends on the previous configuration: the relative displacement uses both frames, and its interpolation weights, tangent basis, and normal force are computed from the previous positions. They remain fixed while solving the current frame, but their dependence on the previous positions is included in the history derivative. With fixed rest geometry,

$$C^{\mathrm{fr},y}_{t,t-1} = \sum_c K_{t,c}^\mathsf{T} \frac{\partial \mathbf{g}^{\mathrm{fr},x}_{t,c}}{\partial \mathbf{x}^c_{t-1}} K_{t-1,c}. \tag{5}$$

Here $K_{t-1,c}$ maps the previous-frame DoFs to the positions of the same contact vertices. Appendix B gives the complete derivation and initial-frame updates.

Algorithm 1 GPU backward pass for a converged trajectory

Input: Converged states $\mathbf{y}_1, \ldots, \mathbf{y}_T$ and frame data; loss partial gradients $\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_T$, $\boldsymbol{\ell}_\theta$
Output: Physical-parameter gradient $\bar{\boldsymbol{\theta}}$

```
b_t ← ℓ_t for all t; θ̄ ← ℓ_θ
for t = T, T − 1, ..., 1 do
    Restore frame t and required earlier-frame data
    Reconstruct affine surface positions and assemble coupled H_t
    Solve H_t^T λ_t = b_t using Appendix Algorithm 2
    If PCG reports failure, stop without returning a parameter gradient
    θ̄ ← θ̄ − G_t^T λ_t                                  ▷ local derivative products
    for s ∈ D(t) do
        b_s ← b_s − C_{t,s}^T λ_t                        ▷ local history products
    end for
end for
return θ̄
```

4.2 Affine rest-shape gradients

We compute rest-shape partial derivatives by varying the rest positions while keeping the FEM positions, affine translations, and affine matrices at their converged values. An affine surface point still changes: $\mathrm{d}\mathbf{x}_{t,i} = A_{t,b}\mathrm{d}\bar{\mathbf{x}}_i$. We include the resulting changes in mass, affine shape energy, collision, and friction.

Mass and shape energy. For rest domain Ω_b and constant density ρ_b, the affine mass block is $M_b = \int_{\Omega_b} \rho_b J(\bar{\mathbf{x}})^\mathsf{T} J(\bar{\mathbf{x}})\,\mathrm{d}V$. We express this volume integral as a sum over the outward-oriented triangles of the closed rest surface. Each triangle and the origin define a signed tetrahedron. Its integrals of ρ_b, $\rho_b\bar{\mathbf{x}}$, and $\rho_b\bar{\mathbf{x}}\bar{\mathbf{x}}^\mathsf{T}$ determine its mass-matrix contribution. Differentiating these expressions with density and connectivity fixed gives $\mathrm{d}M_b$; Appendix Equations (63) and (64) give the formulas. A rest-shape variation then changes the inertial gradient by

$$\mathrm{d}\mathbf{g}^{\mathrm{in}}_{t,b} = (\mathrm{d}M_b)(\mathbf{q}_{t,b} - \widehat{\mathbf{q}}_{t,b}) - M_b\,\mathrm{d}\widehat{\mathbf{q}}_{t,b}. \tag{6}$$

The affine shape energy $h^2\kappa_{\mathrm{A}}V_b\psi_{\mathrm{A}}(\mathbf{q}_{t,b})$ contributes $h^2\kappa_{\mathrm{A}}(\mathrm{d}V_b)\nabla_{\mathbf{q}_{t,b}}\psi_{\mathrm{A}}$ to the energy-gradient change, where V_b is the rest volume and κ_{A} the fixed affine stiffness. Appendix C gives the full derivatives.

Collision and friction. Varying a rest position changes both $\mathbf{x}^c_t$ and $K_{t,c}$. For a scalar rest-position component θ_k, differentiating the collision energy gradient with the simulated DoFs fixed gives

$$\frac{\partial \mathbf{g}^{\mathrm{ct},y}_{t,c}}{\partial \theta_k} = \left(\frac{\partial K_{t,c}}{\partial \theta_k}\right)^\mathsf{T} \mathbf{g}^{\mathrm{ct},x}_{t,c} + K^\mathsf{T}_{t,c} H^{\mathrm{ct},x}_{t,c} \frac{\partial \mathbf{x}^c_t}{\partial \theta_k}. \tag{7}$$

This vector is the collision term's contribution to column k of G_t. A rest-shape variation also changes the contact positions at both t and $t-1$. The resulting friction energy-gradient change in world space is

$$\mathrm{d}\mathbf{g}^{\mathrm{fr},x}_{t,c} = H^{\mathrm{fr},x}_{t,c}\mathrm{d}\mathbf{x}^c_t + C^{\mathrm{fr},x}_{t,c}\mathrm{d}\mathbf{x}^c_{t-1}, \qquad C^{\mathrm{fr},x}_{t,c} = \frac{\partial \mathbf{g}^{\mathrm{fr},x}_{t,c}}{\partial \mathbf{x}^c_{t-1}}. \tag{8}$$

Here $H^{\mathrm{fr},x}_{t,c} = \partial \mathbf{g}^{\mathrm{fr},x}_{t,c}/\partial \mathbf{x}^c_t$. The second term includes changes in the interpolation weights, tangent basis, and normal force computed from the previous positions. Differentiating $K^\mathsf{T}_{t,c}\mathbf{g}^{\mathrm{fr},x}_{t,c}$ also includes the map change $\mathrm{d}K_{t,c}$. If the loss is evaluated on affine surface positions, it has the direct rest-position derivative $A^\mathsf{T}_{t,b}\nabla_{\mathbf{x}_{t,i}}L$ at fixed affine coordinates. This loss derivative contributes to $\boldsymbol{\ell}_\theta$; derivatives of the frame equations contribute to G_t.

Why the shape terms matter. Equation (7) includes both the change in the point map and the change in the world-space collision gradient. Differentiating only the point positions omits the first contribution. Likewise, keeping M_b fixed omits the mass-distribution change in Equation (6). We include these parameter derivatives without adding motion DoFs to the frame system.

4.3 Initial states and joint controls

With rest geometry and mass fixed, the first-frame predictor depends on both the initial configuration and velocity. The second-frame BDF1 predictor also depends directly on the initial configuration. The first- and second-frame inertial derivatives contribute to G_1 and G_2, respectively. First-frame friction can add an initial-configuration derivative. These are parameter-gradient contributions, not a separate frame-zero adjoint. Appendix C.5 gives the formulas and the velocity-reset case.

For a revolute or prismatic joint, targets and conservative efforts enter an actuation energy while its geometric constraints remain unchanged. For an applied scalar control $u_{j,t}$, we accumulate $\bar{u}_{j,t} \mathrel{-}= (\partial \mathbf{g}^{\mathrm{joint}}_{t,j}/\partial u_{j,t})^\mathsf{T}\boldsymbol{\lambda}_t$. The parameter derivative can be nonzero only for the joint's two bodies; its product uses their entries of the coupled frame adjoint. For example, a prismatic joint with displacement ξ, applied target a, stiffness k, and conservative effort f contributes $\Phi^{\mathrm{pri}} = \frac{1}{2}k(\xi - a)^2 - f\xi$. The partial derivatives of its energy gradient with respect to a and f are $-k\nabla_{\mathbf{y}_t}\xi$ and $-\nabla_{\mathbf{y}_t}\xi$, respectively.

Limited targets couple consecutive frames. The applied target $\widetilde{u}_{j,t}$ is clipped against the previous *simulated joint coordinate* rather than the preceding command, with angle wrapping for revolute joints. When the limit is active, the target's gradient therefore flows to the previous frame's right-hand side through the recorded limiter branch; a control shared across joints or frames sums the corresponding contributions. Appendix C.6 gives the limiter equations and the full revolute and geometric derivatives.

5 GPU-PARALLEL BACKWARD AND REUSABLE INTERFACE

Restoring and assembling a frame. After each forward step, we save its converged FEM positions and velocities, affine coordinates and velocities, collision and friction data, and joint-control state. The reverse pass restores these values and the required earlier-frame data. It then reconstructs affine surface positions and evaluates local derivatives without repeating the nonlinear forward solve. We assemble H_t using StiffGIPC's global assembly and symmetric 3×3 sparse block format (Huang et al., 2025a). Appendices D.2.1 and C.6 specify the solver and joint blocks.

CUDA Graphs for adjoint iteration. We solve $H_t^\mathsf{T}\boldsymbol{\lambda}_t = \mathbf{b}_t$ using PCG. Compact affine coordinates reduce the computation within each iteration, but a conventional GPU implementation still launches multiple kernels for matrix products, preconditioning, vector updates, and reductions. For small affine-body systems, kernel-launch overhead can dominate the numerical work, limiting the benefit of reducing DoFs. Checking convergence on the CPU adds synchronization and device-to-host communication at every iteration.

We address both costs with a conditional CUDA Graph that contains PCG initialization and the iteration loop. Rather than submitting each kernel separately from the CPU, we launch the graph and let the GPU repeat the iteration body until convergence or the iteration limit. This reduces repeated host-side launch overhead and eliminates per-iteration convergence transfers, while allowing the iteration count to vary between frames. We reuse the graph across frames with compatible launch configurations and work buffers, amortizing graph construction and instantiation while updating the matrix for each frame. This execution strategy complements affine reduction: compact coordinates reduce numerical work, and graph execution reduces the overhead of repeatedly scheduling that work. We retain the prescribed stopping criteria and directly recompute $\mathbf{b}_t - H_t^\mathsf{T}\boldsymbol{\lambda}_t$ before accepting the adjoint. Appendix Algorithm 2 gives the complete iteration, residual verification, and restart procedure.

Parallel gradient accumulation. Compact affine coordinates reduce the adjoint-system size, but detailed surfaces can still require many local collision and friction derivatives. Once $\boldsymbol{\lambda}_t$ is available, the parameter and earlier-frame updates need only the products $G_t^\mathsf{T}\boldsymbol{\lambda}_t$ and $C_{t,s}^\mathsf{T}\boldsymbol{\lambda}_t$. We therefore evaluate each energy's local derivative contributions in parallel over elements, bodies, joints, and contacts, multiply them by the corresponding adjoint entries, and accumulate the results directly into $\bar{\boldsymbol{\theta}}$ and the affected entries of $\mathbf{b}_s$, without storing or reading back assembled G_t and $C_{t,s}$ matrices (Appendix D.3). All products use the coupled frame adjoint, so this local evaluation does not replace the coupled backward system with independent per-object systems.

Connecting to an optimizer. The GPU backward pass is task-independent: it consumes only the per-frame loss gradients $\boldsymbol{\ell}_t$ and the direct term $\boldsymbol{\ell}_\theta$, and returns the physical-parameter gradient $\bar{\boldsymbol{\theta}}$. Everything specific to an inverse problem is defined outside this pass, in Python: optimizer variables $\mathbf{z}$, a parameter map $\boldsymbol{\theta} = \phi(\mathbf{z})$, and an output map $\mathbf{o} = \mathcal{O}(\mathbf{y}_1, \ldots, \mathbf{y}_T, \boldsymbol{\theta})$ feeding a scalar loss. PyTorch differentiates these definitions to supply $\boldsymbol{\ell}_t$ and $\boldsymbol{\ell}_\theta$, and applies the map derivative to the returned gradient, so the optimizer receives $(\partial\phi/\partial\mathbf{z})^\mathsf{T}\bar{\boldsymbol{\theta}}$ with respect to its own variables. The physical derivatives are therefore implemented once, and changing what is optimized or measured requires no task-specific backward code. New tasks reuse the backward solver when their parameter and output derivatives are supported; Appendix D.4.3 states the supported combinations.

6 EXPERIMENTS

We evaluate ten tasks in motion, material identification, friction optimization, rest-shape design, placement, and articulated control. DiffGIPC runs on an NVIDIA GeForce RTX 4090 GPU; five FEM tasks also run complete optimizations with DiffIPC on a Intel Core i9-13900K CPU. Both systems use double-precision floating-point arithmetic. A forward evaluation simulates a trajectory; a backward evaluation computes its parameter gradient. Appendix E gives the task details, full tables, and settings.

6.1 SHARED FULL-SPACE TASKS

The five shared tasks use FEM in both systems. Figure 2 shows their initial and optimized configurations. Bunny collision optimizes the first body's shared initial velocity, but its loss uses only the second body's volume-weighted horizontal center after $T = 40$ frames, with $L_{\text{bunny}} = \|\mathbf{c}_{xz}(\mathbf{x}_T^{(2)}) - \mathbf{c}_{xz}^*\|^2$. Changing the first body's velocity affects this loss by changing the collision and the second body's resulting motion. Tentacle matching instead uses every output frame. With tetrahedral weights ω_i^{tet} and weighted position $\mathbf{c}_t^{\text{tent}} = \sum_i \omega_i^{\text{tet}} \mathbf{x}_{t,i}$, its objective is $L_{\text{tentacle}} = 20h \sum_{t=1}^{100} \|\mathbf{c}_t^{\text{tent}} - \mathbf{c}_t^{\text{tent},*}\|^2$. The bunny loss supplies a gradient only at the final frame; the tentacle loss supplies one at every frame. Both use the same backward pass, including the tentacle task's friction-history derivatives.

Loaded Support optimizes selected rest vertices of a FEM model under the power-2 stress objective over 20 frames. Material fitting matches loaded-cube markers using two positive stored elastic coefficients. Their parameter map is given in Appendix Equation (103). Its reported loss measures marker-position error. Friction optimization adjusts friction coefficient so that the bunny reaches a prescribed position at a specified time. The loss measures position error at frame $T = 100$, with $h = 0.02\,\text{s}$. The remaining settings are in Appendix Table 4.

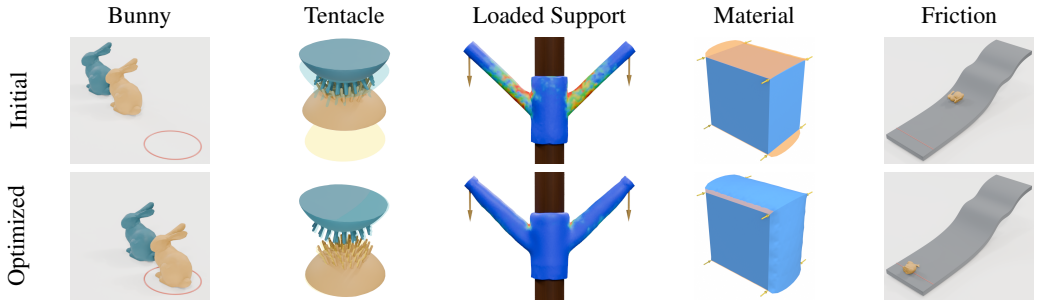


Figure 2: Initial (top) and optimized (bottom) results at frames 40, 100, 20, 19, and 100, respectively. Loaded Support colors show stress; translucent surfaces show targets where present.

Figure 3 reports the loss histories of the five tasks. DiffGIPC attains lower terminal losses than DiffIPC on bunny collision and tentacle matching, and comparable terminal losses on the remaining tasks. The Loaded Support history exhibits intermittent upward jumps: during rest-shape optimization, the support is remeshed periodically to prevent degeneration of its tetrahedral elements, and each remeshing event redefines the discrete objective, producing an abrupt change in the recorded loss. Since both systems converge to comparable terminal losses, the two implementations attain similar optimization quality on these tasks, which provides a numerical basis for the per-evaluation time comparison in Figure 4.

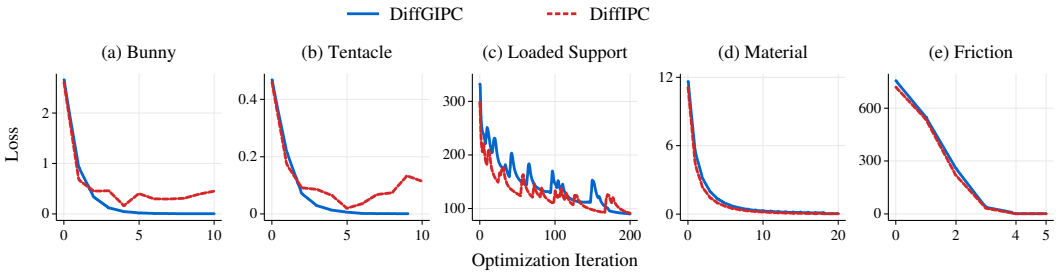


Figure 3: The loss histories for the tasks are presented for qualitative comparison. The horizontal axis in each panel represents the number of accepted optimization updates.

Figure 4 reports 9.09–25.75× speedups for backward evaluation and 10.61–49.45× for forward-plus-backward evaluation relative to DiffIPC. The combined times include the StiffGIPC-based for-

ward simulation. These speedups are computed from mean evaluation times under the reported configurations. Appendix Table 5 gives the phase breakdown.

6.2 Representation and resolution

The bunny study further compares FEM+FEM, ABD+FEM, and ABD+ABD within DiffGIPC, together with DiffIPC FEM+FEM. Each representation generates its own target; these comparisons change the physical representation as well as the coordinate count. As resolution grows from 933 to 28,435 vertices, mean forward-plus-backward time increases from 1.066 to 10.192 s for DiffGIPC FEM+FEM and from 0.181 to 0.519 s for ABD+ABD (Figure 5). The affine surface still incurs collision and derivative work despite its constant number of motion DoFs.

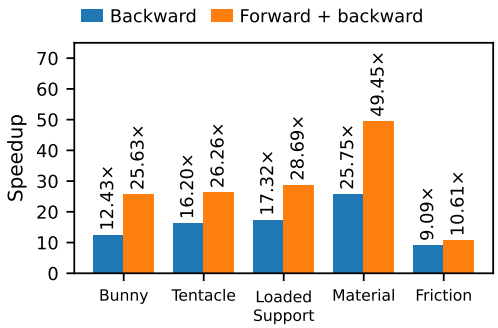


Figure 4: Measured evaluation-time speedups over DiffIPC on five FEM tasks. Appendix Table 5 gives the full measurements.

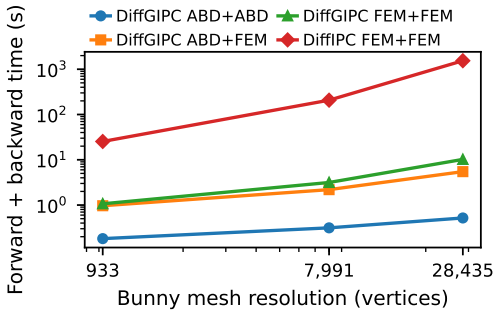


Figure 5: Mean forward-plus-backward time in the bunny resolution study (Appendix Table 6). Both axes are logarithmic.

At the highest resolution, using ABD+ABD instead of FEM+FEM in DiffGIPC gives a $19.65\times$ speedup in paired evaluation; the backward linear phase falls from 2.15 to 0.004 s, while assembly and gradient accumulation account for the remaining backward cost. At this resolution, DiffGIPC FEM+FEM achieves a $151.14\times$ forward-plus-backward speedup over DiffIPC FEM+FEM. DiffGIPC ABD+ABD achieves a $2970.31\times$ speedup over the same baseline, but this comparison combines GPU optimizations and representation changes.

The Garment Hanger representation study retains 4,476 surface vertices and 8,952 triangles. Its FEM replacement adds 902 interior vertices and 18,256 tetrahedra. Mean forward-plus-backward times are 8.511 s for ABD and 14.844 s for FEM, giving ABD a speedup of approximately $1.74\times$. FEM permits non-affine deformation, and the material definitions, rest updates, and volume-mesh quality differ. The result characterizes these implemented models rather than coordinate count alone. Appendix Table 6 reports all phase measurements.

6.3 Design, placement, and articulated control

Shape and placement. Stone placement changes seven bodies' horizontal initial positions, not their shapes; the bodies have 84 motion DoFs. Its differentiable-simulation optimization uses a centroid tolerance of 0.01 in the loss for the first 50 updates and 0.15 afterward, finishing after 78 updates. Cloth fitting optimizes one rest-scale parameter. Two Bézier shoulder scales define the rest shape of a garment hanger represented as an affine body. The support objective depends on the cloth motion, which is coupled to the hanger by collision and friction with coefficient 0.05. Optimization uses the affine mass, collision, and friction shape derivatives and selects scales $(0.8285, 0.8000)$. Thirty optimization rollouts, which may stop when the garment drops, are accompanied by two full 600-frame checks. Appendix E.3 explains the completion and offset used for its progress curve.

Articulated control. Cup catching optimizes joint targets to track plate position, velocity, and affine orientation. The cup is used to check catch success, not as a loss input. The measured run uses 240 frames and terminates at a successful catch after 40 backward evaluations. Dual-arm grasping optimizes one aperture shared by four prismatic joints over the first 142 frames, followed

by a 300-frame grasp check. The later folding controller remains fixed. Figure 6 shows initial and optimized views; their settings are specified in the appendix.

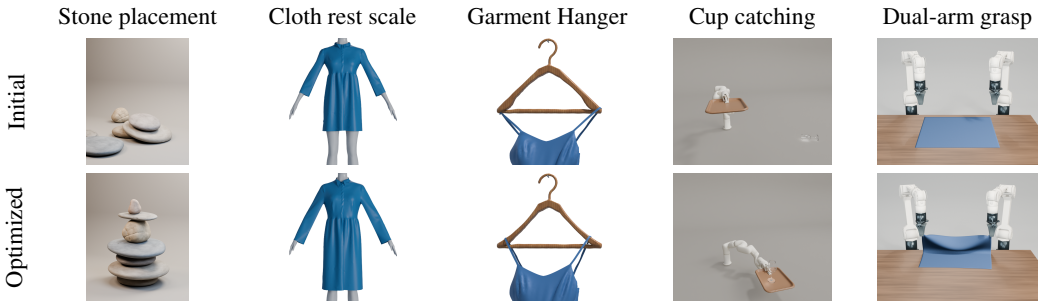


Figure 6: Initial (top) and optimized (bottom) results for five design, placement, and control tasks. Each pair uses the same view and crop. Settings and full sequences are in Appendix E.

Across these tasks, accumulated evaluation times range from 1.32 to 406.77 s. These totals include different call counts and horizons. Table 1 summarizes these tasks; Appendix Table 7 includes the full phase breakdown. Complete optimizations therefore finish in seconds to minutes on a single GPU, even with hundreds of evaluations, demonstrating the efficiency of the framework for repeated gradient-based optimization.

Table 1: DiffGIPC workloads for ten tasks; Appendix Table 7 also includes grasp-angle control. N_f, N_b count forward and backward trajectory evaluations; $\bar{t}_f, \bar{t}_b$ are mean times and T_{eval} their accumulated time in seconds, including trials and checks but excluding setup and export. Horizons can differ.

Shared DiffIPC tasks						Additional tasks					
Task	N_f	N_b	$\bar{t}_f$	$\bar{t}_b$	T_{eval}	Task	N_f	N_b	$\bar{t}_f$	$\bar{t}_b$	T_{eval}
Bunny collision	14	11	0.791	0.275	14.09	Stone placement	80	78	0.370	0.402	60.91
Tentacle matching	11	10	12.754	1.314	153.43	Garment Hanger	32	30	8.120	0.391	271.57
Loaded Support	233	200	0.959	0.916	406.77	Cloth rest scale	7	6	20.654	0.627	148.34
Material fitting	21	20	0.028	0.037	1.32	Cup catching	41	40	5.418	2.139	307.72
Friction optimization	5	5	7.046	0.619	38.33	Dual-arm grasp	11	5	12.789	2.733	154.35

7 Discussion and Conclusion

DiffGIPC separates detailed design parameters from compact motion coordinates by differentiating shape-dependent mass and interaction maps within a coupled GPU adjoint. Together with a reusable interface, this formulation supports heterogeneous inverse problems and reduces recorded simulation and differentiation costs.

The computational gains entail modeling and differentiation limits. Affine bodies cannot resolve local deformation; components requiring it must retain FEM coordinates. Gradients follow recorded contact and control branches, and inactive barriers provide no interaction gradient, limiting guidance toward new contacts. Remeshing is not differentiated, and checkpoint storage grows with trajectory length. Supported parameter combinations remain restricted (Appendix D.4.3). These limits motivate contact-aware search and recomputation-based checkpointing as extensions to the present solver. The performance results also combine implementation and, in the affine comparisons, representation changes; they do not isolate the benefit of each GPU component.

AI USE STATEMENT

AI assisted with polishing and grammar checking.

REFERENCES

Jernej Barbič and Doug L. James. Real-time subspace integration for st. venant-kirchhoff deformable models. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pp. 982–990, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 9781450378253. doi: 10.1145/1186822.1073300. URL https://doi.org/10.1145/1186822.1073300.

Filipe de A. Belbute-Peres, Kevin A. Smith, Kelsey R. Allen, Joshua B. Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 7178–7189, Red Hook, NY, USA, 2018. Curran Associates Inc.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4), July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601116. URL https://doi.org/10.1145/2601097.2601116.

Christopher Brandt, Elmar Eisemann, and Klaus Hildebrandt. Hyper-reduced projective dynamics. *ACM Trans. Graph.*, 37(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201387. URL https://doi.org/10.1145/3197517.3201387.

Youshuai Cai, Chen Li, Haichuan Song, Youchen Xie, and ChangBo Wang. DiffQN: Differentiable Quasi-Newton Method for Elastodynamics. In Marc Christie, Ping-Hsuan Han, Shih-Syun Lin, Nico Pietroni, Teseo Schneider, Hsin-Ruey Tsai, Yu-Shuen Wang, and Eugene Zhang (eds.), *Pacific Graphics Conference Papers, Posters, and Demos*. The Eurographics Association, 2025. ISBN 978-3-03868-295-0. doi: 10.2312/pg.20251265.

Jiong Chen, Florian Schäfer, Jin Huang, and Mathieu Desbrun. Multiscale cholesky preconditioning for ill-conditioned problems. *ACM Trans. Graph.*, 40(4), July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459851. URL https://doi.org/10.1145/3450626.3459851.

Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, Volume 13 - 2019, 2019. ISSN 1662-5218. doi: 10.3389/fnbot.2019.00006. URL https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2019.00006.

Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics. *ACM Trans. Graph.*, 41(2), nov 2021. ISSN 0730-0301. doi: 10.1145/3490168. URL https://doi.org/10.1145/3490168.

Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M. Kaufman, and Daniele Panozzo. Intersection-free rigid body dynamics. *ACM Trans. Graph.*, 40(4), July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459802. URL https://doi.org/10.1145/3450626.3459802.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax – a differentiable physics engine for large scale rigid body simulation, 2021. URL https://arxiv.org/abs/2106.13281.

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 Trans. Graph.*, 39(6), November 2020. ISSN 0730-0301. doi: 10.1145/3414685.3417766. URL https://doi.org/10.1145/3414685.3417766.

Arvi Gjoka, Espen Knoop, Moritz Bächer, Denis Zorin, and Daniele Panozzo. Soft pneumatic actuator design using differentiable simulation. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24*, SIGGRAPH Conference Papers '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705250. doi: 10.1145/3641519.3657467. URL https://doi.org/10.1145/3641519.3657467.

David Hahn, Pol Banzet, James M. Bern, and Stelian Coros. Real2sim: visco-elastic parameter estimation from dynamic motion. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356548. URL https://doi.org/10.1145/3355089.3356548.

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. *Proceedings of 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. *ICLR*, 2020.

Kemeng Huang, Floyd M. Chitalu, Huancheng Lin, and Taku Komura. Gipc: Fast and stable gauss-newton optimization of ipc barrier energy. *ACM Trans. Graph.*, 43(2), March 2024a. ISSN 0730-0301. doi: 10.1145/3643028. URL https://doi.org/10.1145/3643028.

Kemeng Huang, Xinyu Lu, Huancheng Lin, Taku Komura, and Minchen Li. Stiffgipc: Advancing gpu ipc for stiff affine-deformable simulation. *ACM Trans. Graph.*, 44(3), May 2025a. ISSN 0730-0301. doi: 10.1145/3735126. URL https://doi.org/10.1145/3735126.

Zizhou Huang, Daniele Panozzo, and Denis Zorin. Optimized shock-protecting microstructures. *ACM Trans. Graph.*, 43(6), November 2024b. ISSN 0730-0301. doi: 10.1145/3687765. URL https://doi.org/10.1145/3687765.

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 Trans. Graph.*, 43(3), may 2024c. ISSN 0730-0301. doi: 10.1145/3657648. URL https://doi.org/10.1145/3657648.

Zizhou Huang, Maxwell Paik, Zachary Ferguson, Daniele Panozzo, and Denis Zorin. Geometric contact potential. *ACM Trans. Graph.*, 44(4), July 2025b. ISSN 0730-0301. doi: 10.1145/3731142. URL https://doi.org/10.1145/3731142.

Krishna Murthy Jatavallabhula, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jerome Parent-Levesque, 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. URL https://openreview.net/forum?id=c_E8kFWfhp0.

Shih-Yu Lai, Sung-Han Tien, Jui-I Huang, Yen-Chen Tseng, Yi-Ting Chiu, Siyuan Luo, Ziqiu Zeng, Fan Shi, Peter Yichen Chen, Tiantian Liu, Yu-Lun Liu, and Bing-Yu Chen. Diffphd: A unified differentiable solver for projective heterogeneous materials in elastodynamics with contact-rich gpu-acceleration, 2026. URL https://arxiv.org/abs/2605.14526.

Lei Lan, Danny M. Kaufman, Minchen Li, Chenfanfu Jiang, and Yin Yang. Affine body dynamics: fast, stable and intersection-free simulation of stiff materials. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530064. URL https://doi.org/10.1145/3528223.3530064.

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 Trans. Graph. (SIGGRAPH)*, 39(4), 2020.

Xuan Li, Chang Yu, Wenxin Du, Ying Jiang, Tianyi Xie, Yunuo Chen, Yin Yang, and Chenfanfu Jiang. Dress-1-to-3: Single image to simulation-ready 3d outfit with diffusion prior and differentiable physics. *ACM Trans. Graph.*, 44(4), July 2025. ISSN 0730-0301. doi: 10.1145/3731177. URL https://doi.org/10.1145/3731177.

Yifei Li, Tao Du, Kui Wu, Jie Xu, and Wojciech Matusik. Diffcloth: Differentiable cloth simulation with dry frictional contact. *ACM Trans. Graph.*, 42(1), oct 2022. ISSN 0730-0301. doi: 10.1145/3527660. URL https://doi.org/10.1145/3527660.

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

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph.*, 36(3), May 2017. ISSN 0730-0301. doi: 10.1145/2990496. URL https://doi.org/10.1145/2990496.

Mickaël Ly, Jean Jouve, Laurence Boissieux, and Florence Bertails-Descoubes. Projective dynamics with dry frictional contact. *ACM Trans. Graph.*, 39(4), August 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392396. URL https://doi.org/10.1145/3386569.3392396.

Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *ICML*, 2020.

Junior Rojas, Eftychios Sifakis, and Ladislav Kavan. Differentiable implicit soft-body physics, 2021. URL https://arxiv.org/abs/2102.05791.

Hyung Ju Terry Suh, Tao Pang, and Russ Tedrake. Bundled gradients through contact via randomized smoothing. *IEEE Robotics and Automation Letters*, 7(2):4000–4007, 2022. doi: 10.1109/LRA.2022.3146931.

Davi Colli Tozoni, Yunfan Zhou, and Denis Zorin. Optimizing contact-based assemblies. *ACM Transactions on Graphics*, 40(6):269:1–269:19, 2021. ISSN 0730-0301. doi: 10.1145/3478513.3480552. URL https://doi.org/10.1145/3478513.3480552.

Ty Trusty, Otman Benchekroun, Eitan Grinspun, Danny M. Kaufman, and David I.W. Levin. Subspace mixed finite elements for real-time heterogeneous elastodynamics. In *SIGGRAPH Asia 2023 Conference Papers*, SA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703157. doi: 10.1145/3610548.3618220. URL https://doi.org/10.1145/3610548.3618220.

Ty Trusty, Yun (Raymond) Fei, David Levin, and Danny Kaufman. Trading spaces: Adaptive subspace time integration for contacting elastodynamics. *ACM Trans. Graph.*, 43(6), November 2024. ISSN 0730-0301. doi: 10.1145/3687946. URL https://doi.org/10.1145/3687946.

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 *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.034.

Botao Wu, Zhendong Wang, and Huamin Wang. A gpu-based multilevel additive schwarz preconditioner for cloth and deformable body simulation. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530085. URL https://doi.org/10.1145/3528223.3530085.

Xiaohan Ye, Xifeng Gao, Kui Wu, Zherong Pan, and Taku Komura. Sdrs: Shape-differentiable robot simulator. *IEEE Transactions on Robotics*, 2025.

Xiaohan Ye, Kui Wu, Zherong Pan, and Taku Komura. Efficient differentiable contact model with long-range influence. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.20917.

Ziqiu Zeng, Siyuan Luo, Fan Shi, and Zhongkai Zhang. Fast but accurate: A real-time hyperelastic simulator with robust frictional contact. *ACM Trans. Graph.*, 44(4), July 2025. ISSN 0730-0301. doi: 10.1145/3730834. URL https://doi.org/10.1145/3730834.

A Detailed simulation formulation

This appendix expands Section 3: the explicit coordinate order, point maps, dynamic and static equations, local collision and friction laws, single-step derivative, and coupled matrix blocks. One frame is an accepted step of length h in the StiffGIPC-based forward solver (Huang et al., 2025a). Connectivity and vertex-to-body assignments remain fixed during a recorded simulation.

A.1 Simulation degrees of freedom and contact points

A free FEM vertex i is represented by its world-space position $\mathbf{x}_{t,i} \in \mathbb{R}^3$ at frame t. An affine body b is represented by its translation $\mathbf{p}_{t,b} \in \mathbb{R}^3$ and affine transformation $A_{t,b} \in \mathbb{R}^{3\times 3}$. Let $\boldsymbol{\alpha}_{t,b,1}^\mathsf{T}$, $\boldsymbol{\alpha}_{t,b,2}^\mathsf{T}$, and $\boldsymbol{\alpha}_{t,b,3}^\mathsf{T}$ be the rows of $A_{t,b}$. Stacking these vectors with the translation defines the body's twelve DoFs:

$$\begin{aligned} A_{t,b} &= \begin{bmatrix} \boldsymbol{\alpha}_{t,b,1}^\mathsf{T} \\ \boldsymbol{\alpha}_{t,b,2}^\mathsf{T} \\ \boldsymbol{\alpha}_{t,b,3}^\mathsf{T} \end{bmatrix}, \\ \mathbf{q}_{t,b} &= [\mathbf{p}_{t,b}^\mathsf{T}, \boldsymbol{\alpha}_{t,b,1}^\mathsf{T}, \boldsymbol{\alpha}_{t,b,2}^\mathsf{T}, \boldsymbol{\alpha}_{t,b,3}^\mathsf{T}]^\mathsf{T} \in \mathbb{R}^{12}. \end{aligned} \tag{9}$$

For a surface vertex with rest position $\bar{\mathbf{x}}_i$, the world-space position is

$$\mathbf{x}_{t,i} = \mathbf{p}_{t,b} + A_{t,b}\bar{\mathbf{x}}_i. \tag{10}$$

Let I_3 denote the 3×3 identity matrix and $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ the Cartesian unit vectors. With rest position fixed, the Jacobian of this position with respect to $\mathbf{q}_{t,b}$ is the 3×12 matrix

$$\begin{aligned} J_i = J(\bar{\mathbf{x}}_i) &= [I_3 \ \mathbf{e}_1\bar{\mathbf{x}}_i^\mathsf{T} \ \mathbf{e}_2\bar{\mathbf{x}}_i^\mathsf{T} \ \mathbf{e}_3\bar{\mathbf{x}}_i^\mathsf{T}], \\ \mathbf{x}_{t,i} &= J_i\mathbf{q}_{t,b}. \end{aligned} \tag{11}$$

All surface vertices on body b share the same $\mathbf{q}_{t,b}$, but each has its own rest-dependent map J_i.

For N_A affine bodies and N_F free FEM vertices, we collect the simulation DoFs in

$$\mathbf{y}_t = [\mathbf{q}_{t,1}^\mathsf{T}, \ldots, \mathbf{q}_{t,N_\mathrm{A}}^\mathsf{T}, \mathbf{x}_{t,1}^\mathsf{T}, \ldots, \mathbf{x}_{t,N_\mathrm{F}}^\mathsf{T}]^\mathsf{T}. \tag{12}$$

The total number of simulation DoFs is

$$n = 12N_\mathrm{A} + 3N_\mathrm{F}. \tag{13}$$

Adding surface vertices to an affine body can increase contact work, but does not enlarge its block in the linear system. Optimized rest coordinates are design parameters, not additional entries in the frame state. An affine body therefore retains twelve motion DoFs regardless of how many rest coordinates are optimized.

For contact c, we collect the world-space positions of its n_c surface points into $\mathbf{x}_t^c$. Let $\mathbf{y}_t \in \mathbb{R}^n$ contain the free simulation DoFs. The Jacobian $K_{t,c}$ describes how changes in these DoFs affect the contact-point positions:

$$\begin{aligned} \mathbf{x}_t^c &= [\mathbf{x}_{t,i_1}^\mathsf{T}, \ldots, \mathbf{x}_{t,i_{n_c}}^\mathsf{T}]^\mathsf{T}, \\ K_{t,c} &= \frac{\partial \mathbf{x}_t^c}{\partial \mathbf{y}_t} \in \mathbb{R}^{3n_c\times n}. \end{aligned} \tag{14}$$

Each group of three rows corresponds to one contact point. For a free FEM vertex, its three positional DoFs appear directly in $\mathbf{y}_t$, so these rows contain an identity block I_3 in the corresponding columns. For a point on a free affine body, its position depends on the body's twelve affine DoFs through J_i, so these rows contain J_i in that body's columns. For a prescribed point, the rows are zero because its position does not depend on the free DoFs. Entries outside the corresponding FEM or affine-body columns are zero in every row block. When differentiating with respect to the state, we hold the rest geometry and prescribed motion fixed. When differentiating with respect to the rest shape, we also differentiate the dependence of the affine point maps J_i, and hence $K_{t,c}$, on the rest positions (Section C.2).

A.2 INCREMENTAL POTENTIAL AND ITS ENERGY GRADIENT

Let $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$ collect the physical parameters selected for optimization, such as material and friction coefficients, rest coordinates, initial states, and joint controls. Here n_θ is the number of selected scalar parameters. These parameters are held fixed during one forward simulation. A joint-control schedule can still contain different prescribed values for different frames. For a given $\boldsymbol{\theta}$, each time step of length h computes the simulation DoFs $\mathbf{y}_t$ by minimizing the incremental potential

$$\begin{aligned}\Phi_t(\mathbf{y}_t) = \tfrac{1}{2}\|\mathbf{y}_t - \widehat{\mathbf{y}}_t\|_M^2 \\ + \sum_{e\in\mathcal{E}} w_e(h,\boldsymbol{\theta})\Psi_{t,e}(\mathbf{y}_t, \mathbf{y}_{t-1}; \boldsymbol{\theta}).\end{aligned} \tag{15}$$

The first term accounts for inertia, with $\|\mathbf{v}\|_M^2 = \mathbf{v}^\mathsf{T} M\mathbf{v}$. The mass matrix M is expressed in the simulation coordinates: it contains vertex-mass blocks for FEM and a 12×12 block for each affine body. Section C gives the affine mass matrix and its rest-shape derivatives. The predictor $\widehat{\mathbf{y}}_t$ is computed from earlier states and prescribed accelerations $\mathbf{a}_t$. For dynamic bodies, backward Euler (BDF1) gives

$$\widehat{\mathbf{y}}_t = 2\mathbf{y}_{t-1} - \mathbf{y}_{t-2} + h^2\mathbf{a}_t, \qquad t \geq 2, \tag{16}$$

At the first step, we use $\widehat{\mathbf{y}}_1 = \mathbf{y}_0 + h\dot{\mathbf{y}}_0 + h^2\mathbf{a}_1$, where $\mathbf{y}_0$ specifies the initial configuration and $\dot{\mathbf{y}}_0$ contains the initial FEM vertex velocities and affine-coordinate velocities.

The second term in Equation (15) sums the contributions from FEM elastic energy, affine shape energy, collision, friction, and joints. We index these contributions by $e \in \mathcal{E}$ and write each as $w_e(h,\boldsymbol{\theta})\Psi_{t,e}$, where $\Psi_{t,e}$ is the corresponding energy function and $w_e(h,\boldsymbol{\theta})$ is its scalar weight. The collision energy is the non-penetration barrier, distinct from the friction energy for tangential resistance. IPC's friction energy depends on the relative tangential displacement of the two configurations, determined by $\mathbf{y}_t$ and $\mathbf{y}_{t-1}$. Its interpolation weights, tangent bases, and normal forces are computed from the previous-frame positions and held fixed while solving for $\mathbf{y}_t$. Their dependence on those positions is included when differentiating with respect to $\mathbf{y}_{t-1}$.

At a minimum of the incremental potential, its gradient with respect to the free simulation DoFs vanishes. Denoting this energy gradient by $\mathbf{g}_t$, we obtain the stationarity equation

$$\begin{aligned}\mathbf{g}_t(\mathbf{y}_t, \mathbf{y}_{t-1}, \mathbf{y}_{t-2}; \boldsymbol{\theta}) &:= \nabla_{\mathbf{y}_t}\Phi_t \\ &= M(\mathbf{y}_t - \widehat{\mathbf{y}}_t) \\ &\quad + \sum_{e\in\mathcal{E}} w_e(h,\boldsymbol{\theta})\nabla_{\mathbf{y}_t}\Psi_{t,e} \\ &= \mathbf{0}.\end{aligned} \tag{17}$$

The energy gradient $\mathbf{g}_t$ is also the nonlinear residual of this equation. For each parameter choice, the forward solver finds DoFs that satisfy this stationarity condition. We next differentiate this condition to compute how a parameter change affects the solved DoFs and the loss evaluated on them.

Energy scaling. Forward and backward calculations use the same coefficients in each incremental-potential term. Elastic, cloth, bending, and stitch terms use their h^2 scale. The self and ground collision barriers already contain κ; friction uses its configured coefficient and the normal force. The affine shape term is $h^2\kappa_\mathrm{A}V_b\psi_\mathrm{A}(\mathbf{q}_{t,b})$, where V_b is the rest volume, κ_A the affine stiffness, and ψ_A the near-rigidity energy density. Joint-actuation terms use their effective driving coefficients directly. A coefficient already included in a local energy is not applied again during assembly.

A.3 TIME STEPPING AND STATIC-EQUILIBRIUM DERIVATIVES

The dynamic predictors above depend on earlier configurations and velocity. For the velocity-reset approximation, substituting $\widehat{\mathbf{y}}_t = \mathbf{y}_{t-1} + h^2\mathbf{a}_t$ into the step equation gives

$$\begin{aligned}\mathbf{g}_t = M(\mathbf{y}_t - \mathbf{y}_{t-1}) - h^2 M\mathbf{a}_t \\ + \sum_{e\in\mathcal{E}} w_e\nabla_{\mathbf{y}_t}\Psi_{t,e} = \mathbf{0}.\end{aligned} \tag{18}$$

Convergence of the current Newton iteration does not remove $M(\mathbf{y}_t - \mathbf{y}_{t-1})$ from the step equation. We differentiate a finite sequence of these velocity-reset updates with the frame-wise adjoint. For a static equilibrium without active friction, the differentiated configuration instead satisfies force balance under the fixed loads and boundary conditions:

$$-h^2 M\mathbf{a} + \sum_{e\in\mathcal{E}} w_e \nabla_{\mathbf{y}} \Psi_e(\mathbf{y};\boldsymbol{\theta}) = \mathbf{0}, \tag{19}$$

Let H_Ψ be the Hessian contribution from the summed non-inertial energy terms. On the free DoFs, the step and equilibrium matrices are

$$H^{\text{step}} = M + H_\Psi, \qquad H^{\text{eq}} = H_\Psi. \tag{20}$$

The single-step adjoint in Section A.4 applies to either equation, with H and G computed from that equation. Removing the displacement-dependent term does not remove the dependence of gravity on mass. For the supported cloth-rest parameters, G includes membrane and bending derivatives and the gravity contribution $-h^2(\partial M/\partial\theta_k)\mathbf{a}$. This equilibrium derivative is used for the supported pure-FEM and cloth-rest cases. Volumetric rest parameters, active friction, initial velocity, and mixed FEM–affine states are excluded.

A.4 Parameter gradients from the step equation

We first consider a single step and omit the frame index. Any optimized initial-state components are included in $\boldsymbol{\theta}$. The converged DoFs satisfy $\mathbf{g}(\mathbf{y},\boldsymbol{\theta}) = \mathbf{0}$. A parameter change affects this equation both directly and through the resulting change in $\mathbf{y}$. Differentiating the stationarity equation gives

$$\underbrace{\frac{\partial \mathbf{g}}{\partial \mathbf{y}}}_{H} \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\boldsymbol{\theta}} + \underbrace{\frac{\partial \mathbf{g}}{\partial \boldsymbol{\theta}}}_{G} = \mathbf{0}. \tag{21}$$

Here G contains partial derivatives of the energy gradient with respect to the physical parameters, evaluated while the simulated DoFs remain at their converged values. The derivative H with respect to the DoFs is the Hessian of the incremental potential:

$$H = \frac{\partial \mathbf{g}}{\partial \mathbf{y}} = \nabla^2_{\mathbf{y}} \Phi. \tag{22}$$

Both matrices are evaluated at the converged DoFs. Equation (21) therefore gives a linear system with H for computing $\mathrm{d}\mathbf{y}/\mathrm{d}\boldsymbol{\theta}$. Restoring the frame index, we denote these matrices by $H_t \in \mathbb{R}^{n_t \times n_t}$ and $G_t \in \mathbb{R}^{n_t \times n_\theta}$.

For a scalar loss $L(\mathbf{y},\boldsymbol{\theta})$, let $\boldsymbol{\ell}_y = \nabla_{\mathbf{y}} L$ and $\boldsymbol{\ell}_\theta = \nabla_{\boldsymbol{\theta}} L$ denote its partial gradients with respect to the DoFs and physical parameters. In the latter, $\mathbf{y}$ is fixed; the total parameter gradient also includes the change in the solved state. The chain rule gives

$$\nabla_{\boldsymbol{\theta}} L = \boldsymbol{\ell}_\theta + \left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\boldsymbol{\theta}}\right)^{\mathsf{T}} \boldsymbol{\ell}_y. \tag{23}$$

Explicitly computing $\mathrm{d}\mathbf{y}/\mathrm{d}\boldsymbol{\theta}$ becomes costly as the number of optimized parameters grows. We avoid forming this matrix and compute the loss gradient by solving the adjoint system:

$$H^{\mathsf{T}}\boldsymbol{\lambda} = \boldsymbol{\ell}_y. \tag{24}$$

The solution $\boldsymbol{\lambda}$ is the adjoint vector. Combining this solve with Equation (21) gives the parameter gradient

$$\nabla_{\boldsymbol{\theta}} L = \boldsymbol{\ell}_\theta - G^{\mathsf{T}}\boldsymbol{\lambda}. \tag{25}$$

For a single time step, all parameter gradients of a scalar loss share the same adjoint, obtained by solving one coupled linear system. This system retains the collision, friction, and joint couplings among the FEM and affine DoFs. For trajectories, we extend the formulation to account for the dependence of earlier states on the parameters (Section B).

A.5 COLLISION AND FRICTION DERIVATIVES ACROSS REPRESENTATIONS

The adjoint equation (24) requires derivatives with respect to simulation DoFs. Collision and friction energies, however, are evaluated at world-space surface points. We therefore compute their point-space derivatives first and map them to the FEM and affine DoFs using $K_{t,c}$ from Equation (14). For the collision energy, we use the squared-distance barrier

$$b_{\mathrm{bar}}(d) = \begin{cases} \kappa(d-\widehat{d})^2 \log^2(d/\widehat{d}), & 0 < d < \widehat{d}, \\ 0, & d \geq \widehat{d}, \end{cases} \tag{26}$$

where d is the squared distance, $\widehat{d}$ is the squared-distance threshold below which the barrier is active, and κ is the barrier coefficient. At each Newton iteration, we truncate the step along the search direction using continuous collision detection to prevent intersections, followed by line search to ensure a decrease in the incremental potential. The backward pass evaluates derivatives at the converged state of each frame. We therefore save this state and the corresponding collision and friction data when the frame solve finishes.

Barrier derivatives and normal force. For an active collision pair, define $\Delta_d = d - \widehat{d}$ and $\ell_d = \log(d/\widehat{d})$. The barrier derivatives are

$$\begin{aligned} b_{\mathrm{bar}}(d) &= \kappa \Delta_d^2 \ell_d^2, \\ b'_{\mathrm{bar}}(d) &= 2\kappa \Delta_d \ell_d (\ell_d + \Delta_d/d), \\ b''_{\mathrm{bar}}(d) &= 2\kappa \left[\ell_d^2 + 4\Delta_d \ell_d / d + \Delta_d^2 (1 - \ell_d)/d^2 \right]. \end{aligned} \tag{27}$$

For $d = d(\mathbf{x}_t^c)$, the collision gradient and Hessian in world-space point coordinates are

$$\begin{aligned} \mathbf{g}_{t,c}^{\mathrm{ct},x} &= b'_{\mathrm{bar}}(d) \nabla_{\mathbf{x}_t^c} d, \\ H_{t,c}^{\mathrm{ct},x} &= b''_{\mathrm{bar}}(d) (\nabla_{\mathbf{x}_t^c} d)(\nabla_{\mathbf{x}_t^c} d)^{\mathsf{T}} + b'_{\mathrm{bar}}(d) \nabla^2_{\mathbf{x}_t^c} d. \end{aligned} \tag{28}$$

The corresponding normal-force magnitude and its distance derivative are

$$\begin{aligned} f_n &= -2\sqrt{d}\, b'_{\mathrm{bar}}(d), \\ \frac{\partial f_n}{\partial d} &= -\frac{b'_{\mathrm{bar}}(d)}{\sqrt{d}} - 2\sqrt{d}\, b''_{\mathrm{bar}}(d). \end{aligned} \tag{29}$$

The barrier, normal force, and their derivatives already include the coefficient κ. For the friction formulas below, the fixed frame and contact indices are omitted on the normal force, interpolation weights, and tangent basis.

Smoothed friction energy. For a contact with n_c points, let $W = [w_1^{\mathrm{fr}} I_3 \ \cdots \ w_{n_c}^{\mathrm{fr}} I_3]$ combine the vertex displacements into the relative displacement of the two sides. The signed closest-point interpolation weights satisfy $\sum_i w_i^{\mathrm{fr}} = 0$ and differ from the energy weights w_e. Let $U \in \mathbb{R}^{3\times 2}$ contain the tangent basis vectors. The relative tangential displacement and its squared norm are

$$\boldsymbol{\xi} = U^{\mathsf{T}} W (\mathbf{x}_t^c - \mathbf{x}_{t-1}^c), \qquad \sigma = \|\boldsymbol{\xi}\|^2. \tag{30}$$

Point–point, point–edge, point–triangle, and edge–edge features use their corresponding weights and bases. With smoothing length ϵ_{fr}, the self-friction law is

$$\varphi_{\mathrm{self}}(\sigma) = \begin{cases} \dfrac{\sigma(-\sqrt{\sigma}/3 + \epsilon_{\mathrm{fr}})}{\epsilon_{\mathrm{fr}}^2} + \dfrac{\epsilon_{\mathrm{fr}}}{3}, & \sigma \leq \epsilon_{\mathrm{fr}}^2, \\ \sqrt{\sigma}, & \sigma > \epsilon_{\mathrm{fr}}^2, \end{cases} \tag{31}$$

and the ground-friction law is

$$\varphi_{\mathrm{ground}}(\sigma) = \begin{cases} \sigma/(2\epsilon_{\mathrm{fr}}), & \sigma \leq \epsilon_{\mathrm{fr}}^2, \\ \sqrt{\sigma} - \epsilon_{\mathrm{fr}}/2, & \sigma > \epsilon_{\mathrm{fr}}^2. \end{cases} \tag{32}$$

The configured smoothing input is used as $\epsilon_{\mathrm{fr}} = h\sqrt{\widehat{d}_f}$. This relation is retained as implemented; the physical units of the configuration input $\widehat{d}_f$ are not specified by the experimental record. With friction coefficient μ, the energy is $\mu f_n \varphi(\sigma)$, and its point-space gradient is

$$\mathbf{g}_{t,c}^{\mathrm{fr},x} = \mu f_n W^{\mathsf{T}} U \, \nabla_{\boldsymbol{\xi}} \varphi(\|\boldsymbol{\xi}\|^2). \tag{33}$$

The notation $\nabla_{\boldsymbol{\xi}} \varphi(\|\boldsymbol{\xi}\|^2)$ denotes the gradient with respect to the two tangential-displacement components, including the dependence of $\|\boldsymbol{\xi}\|^2$ on them. For self friction below the smoothing threshold,

$$\nabla_{\boldsymbol{\xi}} \varphi_{\mathrm{self}}(\|\boldsymbol{\xi}\|^2) = \frac{2\epsilon_{\mathrm{fr}} - \|\boldsymbol{\xi}\|}{\epsilon_{\mathrm{fr}}^2} \boldsymbol{\xi}. \tag{34}$$

Above the threshold, both laws give $\boldsymbol{\xi}/\|\boldsymbol{\xi}\|$; ground friction below the threshold gives $\boldsymbol{\xi}/\epsilon_{\mathrm{fr}}$. The resulting 2×2 tangential Hessian is mapped to point coordinates using W and U.

Mapping local derivatives to simulation coordinates. For a contact c at frame t, let $\mathbf{g}_{t,c}^{x}$ and $H_{t,c}^{x}$ denote the gradient and Hessian of the collision energy with respect to the stacked world-space point positions $\mathbf{x}_t^c$. The superscript x denotes point coordinates, whereas y denotes simulation DoFs. With rest geometry fixed, the point map $K_{t,c}$ from Equation (14) is constant with respect to $\mathbf{y}_t$. The chain rule therefore gives the contributions to the frame energy gradient and Hessian:

$$\mathbf{g}_{t,c}^{y} = K_{t,c}^{\mathsf{T}} \mathbf{g}_{t,c}^{x}, \qquad H_{t,c}^{y} = K_{t,c}^{\mathsf{T}} H_{t,c}^{x} K_{t,c}. \tag{35}$$

The same transformation applies to friction derivatives with respect to the current-frame DoFs. Friction derivatives with respect to earlier states are included in the trajectory calculation in Section B.

Each point uses its FEM or affine coordinate map. For point i on affine body b and point j on affine body b', let H_{ij}^{x} be their 3×3 point-space Hessian block. Mapping this block gives a 12×12 contribution to $H_{bb'}^{\mathrm{AA}}$. If the second point is instead a FEM vertex j, its identity map gives a 12×3 contribution to H_{bj}^{AF}:

$$\begin{aligned} H_{bb'}^{\mathrm{AA}} &\mathrel{+}= J_i^{\mathsf{T}} H_{ij}^{x} J_j, \\ H_{bj}^{\mathrm{AF}} &\mathrel{+}= J_i^{\mathsf{T}} H_{ij}^{x}. \end{aligned} \tag{36}$$

All surface vertices of an affine body share its twelve DoFs. Contributions from different vertices of that body are therefore summed into the same body blocks, rather than assigned independent vertex DoFs. For distinct points on the same body, both ordered point-block contributions must be included when they map to the same diagonal body block.

After assembly, grouping affine (A) and finite-element (F) DoFs gives the frame Hessian

$$H_t = \begin{bmatrix} H_{\mathrm{AA}} & H_{\mathrm{AF}} \\ H_{\mathrm{FA}} & H_{\mathrm{FF}} \end{bmatrix}, \qquad H_{\mathrm{FA}} = H_{\mathrm{AF}}^{\mathsf{T}}. \tag{37}$$

The blocks H_{AA} and H_{FF} contain derivatives within the affine and FEM coordinates, respectively. The blocks H_{AF} and H_{FA} contain the cross derivatives produced by collision and friction between the two representations. Joint terms also couple the DoFs of the two connected affine bodies within H_{AA}. These blocks describe how changing one object's DoFs changes the energy gradient on another object. Compact affine coordinates thus reduce the matrix dimension without removing interactions between objects. The adjoint equation must therefore retain these cross blocks, so the loss gradient can account for interactions between objects. Figure 7 summarizes the point-to-DoF mapping and the coupled Hessian structure in Equation (37).

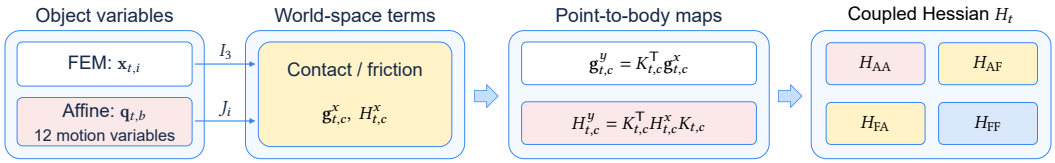


Figure 7: From world-space collision and friction derivatives to the coupled Hessian H_t. FEM vertices use I_3 and affine points use J_i; $K_{t,c}$ collects these maps. The contribution $H^y_{t,c} = K^\mathsf{T}_{t,c}H^x_{t,c}K_{t,c}$ follows Equation (35). Both cross blocks H_{AF} and $H_{\mathrm{FA}} = H^\mathsf{T}_{\mathrm{AF}}$ remain in the assembled system. Rest-shape derivatives also differentiate the point maps.

The same point maps form the loss-gradient vector $\boldsymbol{\ell}_y$ on the right-hand side of Equation (24). For a loss evaluated on world-space surface points, its gradient with respect to the affine coordinates of body b, whose surface-vertex index set is $\mathcal{V}_b$, is

$$\nabla_{\mathbf{q}_b} L = \sum_{i \in \mathcal{V}_b} J_i^\mathsf{T} \nabla_{\mathbf{x}_i} L. \tag{38}$$

For a FEM vertex, the loss gradient with respect to its DoFs equals that with respect to its world-space position. These mappings hold with rest geometry fixed; Section C.2 includes the additional dependence of the point maps and surface positions on rest shape. We have now defined the single-step adjoint and the collision and friction terms that couple the two representations. Section B extends this calculation to losses on multiple frames and to equations that depend on earlier states.

B Full Trajectory-Adjoint Derivation

This appendix gives the full derivation underlying Section 4.1, including the three-frame example, initial-state terms, and inertia and friction history products. Algorithm 1 in Section 4.1 summarizes the computation.

B.1 Equations for the Whole Trajectory

Let $\mathbf{y}_t \in \mathbb{R}^{n_t}$ contain the complete simulation DoFs at frame t, with the ordering defined in Section A.1. Unlike the single-step calculation, we now include how earlier solved states change with $\boldsymbol{\theta}$. The frame equation is

$$\mathbf{g}_t(\mathbf{y}_t, \{\mathbf{y}_s : s \in \mathcal{D}(t)\}, \boldsymbol{\theta}) = \mathbf{0}. \tag{39}$$

Here, $\mathcal{D}(t) \subseteq \{1, \ldots, t-1\}$ indexes the earlier frames whose DoFs affect the equation at frame t. These earlier DoFs also depend on $\boldsymbol{\theta}$, so their derivatives must be included when differentiating the trajectory. The initial state is an input, not a solved frame, and is excluded from $\mathcal{D}(t)$. When initial-state components are optimized, their partial derivatives contribute to the corresponding columns of G_t, as do derivatives with respect to selected joint controls.

At each frame, H_t and G_t are the state and parameter derivatives introduced in Section A.4. We also need the derivative with respect to an earlier state:

$$C_{t,s} = \frac{\partial \mathbf{g}_t}{\partial \mathbf{y}_s} \in \mathbb{R}^{n_t \times n_s}, \qquad s \in \mathcal{D}(t). \tag{40}$$

Here, $C_{t,s}$ describes how the DoFs at an earlier frame s affect the energy gradient at frame t, and is zero when $s \notin \mathcal{D}(t)$. Since $\mathbf{y}_s$ also depends on $\boldsymbol{\theta}$, this dependence contributes $C_{t,s}\,\mathrm{d}\mathbf{y}_s/\mathrm{d}\boldsymbol{\theta}$ when differentiating the frame equation with respect to $\boldsymbol{\theta}$. The trajectory derivatives account for both these temporal dependencies and the coupling within each frame: H_t includes all FEM and affine DoFs and retains their interactions through collision, friction, and joints. Differentiating Equation (39) gives

$$H_t \frac{\mathrm{d}\mathbf{y}_t}{\mathrm{d}\boldsymbol{\theta}} + \sum_{s \in \mathcal{D}(t)} C_{t,s} \frac{\mathrm{d}\mathbf{y}_s}{\mathrm{d}\boldsymbol{\theta}} + G_t = 0. \tag{41}$$

Let $g_{t,\alpha} = (\mathbf{g}_t)_\alpha$ denote the α-th component of the energy gradient at frame t. The same relation can then be written componentwise as

$$\sum_{s=1}^{t}\sum_{\beta=1}^{n_s}\frac{\partial g_{t,\alpha}}{\partial y_{s,\beta}}\frac{\mathrm{d}y_{s,\beta}}{\mathrm{d}\theta_k} = -(G_t)_{\alpha k}. \tag{42}$$

Here, $y_{s,\beta}$ denotes the β-th DoF at frame s, and θ_k denotes the k-th parameter.

For compact notation, let $S_t = \mathrm{d}\mathbf{y}_t/\mathrm{d}\boldsymbol{\theta} \in \mathbb{R}^{n_t \times n_\theta}$. Stacking all frames makes the time structure explicit:

$$\underbrace{\begin{bmatrix} H_1 & 0 & \cdots & 0 \\ C_{2,1} & H_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ C_{T,1} & \cdots & C_{T,T-1} & H_T \end{bmatrix}}_{\mathcal{J}} \underbrace{\begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_T \end{bmatrix}}_{\mathcal{S}} = -\underbrace{\begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_T \end{bmatrix}}_{\mathcal{G}}. \tag{43}$$

A block below the diagonal is zero when the corresponding earlier state does not enter that frame's equation. The matrix is therefore block lower triangular, with nonzero history blocks only for the dependencies in $\mathcal{D}(t)$. Although each H_t is a spatial Hessian, $C_{t,s}$ need not be symmetric. The whole matrix $\mathcal{J}$ is the Jacobian of the stacked step equations, not the Hessian of one energy over all frames.

As shown in Equations (16) and (17), our position-based backward Euler (BDF1) formulation makes the equation at frame $t \geq 2$ depend directly on the current state $\mathbf{y}_t$ and the two previous configurations $\mathbf{y}_{t-1}$ and $\mathbf{y}_{t-2}$. The initial configuration is a parameter input, as described above. We illustrate these dependencies using the first three solved frames, for which Equation (43) becomes

$$\begin{bmatrix} H_1 & 0 & 0 \\ C_{2,1} & H_2 & 0 \\ C_{3,1} & C_{3,2} & H_3 \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} = -\begin{bmatrix} G_1 \\ G_2 \\ G_3 \end{bmatrix}. \tag{44}$$

This matrix illustrates the dependencies between frames. The matrices $S_t = \mathrm{d}\mathbf{y}_t/\mathrm{d}\boldsymbol{\theta}$ contain derivatives of the solved frame DoFs with respect to the physical parameters. The adjoint method below computes the loss gradient without explicitly computing these derivative matrices. It solves one coupled system per frame instead of assembling the full-trajectory matrix.

B.2 Reverse-time adjoint equations

We extend the single-step adjoint in Equation (24) to the trajectory loss $L(\mathbf{y}_1, \ldots, \mathbf{y}_T; \boldsymbol{\theta})$. Let $\boldsymbol{\ell}_t = \nabla_{\mathbf{y}_t} L$ denote its partial gradient with respect to the simulation DoFs at frame t, and let $\boldsymbol{\ell}_\theta$ denote its partial parameter gradient with all simulated states fixed. For a loss evaluated only on the final state, $\boldsymbol{\ell}_t = \mathbf{0}$ for $t < T$, although earlier states can still affect the loss through the subsequent motion. Using $S_t = \mathrm{d}\mathbf{y}_t/\mathrm{d}\boldsymbol{\theta}$ from Section B.1, the chain rule extends Equation (23) to

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} L &= \boldsymbol{\ell}_\theta + \sum_{t=1}^{T}\left(\frac{\mathrm{d}\mathbf{y}_t}{\mathrm{d}\boldsymbol{\theta}}\right)^{\mathsf{T}}\boldsymbol{\ell}_t \\ &= \boldsymbol{\ell}_\theta + \sum_{t=1}^{T} S_t^{\mathsf{T}}\boldsymbol{\ell}_t \\ &= \boldsymbol{\ell}_\theta + \mathcal{S}^{\mathsf{T}}\boldsymbol{\ell}_{\mathrm{all}}, \end{aligned} \tag{45}$$

where $\boldsymbol{\ell}_{\mathrm{all}}$ stacks the per-frame loss gradients in the same order as the blocks of $\mathcal{S}$. The first term accounts for the explicit parameter dependence of the loss; the second accounts for changes in the simulated trajectory.

To evaluate the trajectory contribution without computing $\mathcal{S}$, we introduce an adjoint vector $\boldsymbol{\lambda}_t$ for each frame and solve

$$\mathcal{J}^{\mathsf{T}}\underbrace{\begin{bmatrix} \boldsymbol{\lambda}_1 \\ \vdots \\ \boldsymbol{\lambda}_T \end{bmatrix}}_{\boldsymbol{\Lambda}} = \underbrace{\begin{bmatrix} \boldsymbol{\ell}_1 \\ \vdots \\ \boldsymbol{\ell}_T \end{bmatrix}}_{\boldsymbol{\ell}_{\mathrm{all}}}. \tag{46}$$

Combining this equation with $\mathcal{J}\mathcal{S} = -\mathcal{G}$ from Equation (43) gives

$$\mathcal{S}^\mathsf{T}\boldsymbol{\ell}_{\text{all}} = \mathcal{S}^\mathsf{T}\mathcal{J}^\mathsf{T}\boldsymbol{\Lambda} = -\mathcal{G}^\mathsf{T}\boldsymbol{\Lambda}. \tag{47}$$

Thus, the parameter derivatives and adjoints provide the trajectory contribution without explicitly forming $\mathrm{d}\mathbf{y}_t/\mathrm{d}\boldsymbol{\theta}$.

The temporal structure of $\mathcal{J}$ allows the adjoints to be computed one frame at a time. Each frame equation depends only on the current and earlier states, so $\mathcal{J}$ is block lower triangular and $\mathcal{J}^\mathsf{T}$ is block upper triangular. For the three-frame system in Equation (44), Equation (46) becomes

$$\begin{bmatrix} H_1^\mathsf{T} & C_{2,1}^\mathsf{T} & C_{3,1}^\mathsf{T} \\ 0 & H_2^\mathsf{T} & C_{3,2}^\mathsf{T} \\ 0 & 0 & H_3^\mathsf{T} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ \boldsymbol{\lambda}_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\ell}_1 \\ \boldsymbol{\ell}_2 \\ \boldsymbol{\ell}_3 \end{bmatrix}. \tag{48}$$

The last block row determines $\boldsymbol{\lambda}_3$. Substituting this result into the second row gives $\boldsymbol{\lambda}_2$, after which the first row determines $\boldsymbol{\lambda}_1$:

$$\begin{aligned} H_3^\mathsf{T}\boldsymbol{\lambda}_3 &= \boldsymbol{\ell}_3, \\ H_2^\mathsf{T}\boldsymbol{\lambda}_2 &= \boldsymbol{\ell}_2 - C_{3,2}^\mathsf{T}\boldsymbol{\lambda}_3, \\ H_1^\mathsf{T}\boldsymbol{\lambda}_1 &= \boldsymbol{\ell}_1 - C_{2,1}^\mathsf{T}\boldsymbol{\lambda}_2 - C_{3,1}^\mathsf{T}\boldsymbol{\lambda}_3. \end{aligned} \tag{49}$$

Applying the same backward substitution to all T frames gives

$$H_t^\mathsf{T}\boldsymbol{\lambda}_t = \boldsymbol{\ell}_t - \sum_{\substack{\tau > t \\ t \in \mathcal{D}(\tau)}} C_{\tau,t}^\mathsf{T}\boldsymbol{\lambda}_\tau, \qquad t = T, \ldots, 1. \tag{50}$$

The sum includes only later frames whose equations depend on $\mathbf{y}_t$. Once the adjoints are known, Equations (45) and (47) give the parameter gradient:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} L &= \boldsymbol{\ell}_\theta - \mathcal{G}^\mathsf{T}\boldsymbol{\Lambda} \\ &= \boldsymbol{\ell}_\theta - \sum_{t=1}^{T} G_t^\mathsf{T}\boldsymbol{\lambda}_t. \end{aligned} \tag{51}$$

B.3 Computing one coupled system per frame

We evaluate Equations (50) and (51) in a single reverse pass. Let $\mathbf{b}_t$ accumulate the right-hand side for frame t, and let $\bar{\boldsymbol{\theta}}$ accumulate the parameter gradient. Before processing the frames, we initialize $\mathbf{b}_t = \boldsymbol{\ell}_t$ for $t = 1, \ldots, T$ and $\bar{\boldsymbol{\theta}} = \boldsymbol{\ell}_\theta$. We then visit frames from T to 1. At each frame t, we restore the converged state recorded by the forward simulation, assemble the coupled Hessian H_t as described in Section A.5, and solve

$$H_t^\mathsf{T}\boldsymbol{\lambda}_t = \mathbf{b}_t. \tag{52}$$

Using this adjoint, we update the earlier-frame right-hand sides and accumulate the parameter gradient:

$$\mathbf{b}_s \mathrel{-}= C_{t,s}^\mathsf{T}\boldsymbol{\lambda}_t, \qquad s \in \mathcal{D}(t), \tag{53}$$

$$\bar{\boldsymbol{\theta}} \mathrel{-}= G_t^\mathsf{T}\boldsymbol{\lambda}_t. \tag{54}$$

Each update to $\mathbf{b}_s$ comes from a later frame $t > s$ and is therefore applied before frame s is solved. Consequently, each solve uses the complete right-hand side in Equation (50).

To evaluate Equations (53) and (54), we compute each energy's contributions to $C_{t,s}$ and G_t. We multiply the transposes of these local derivative blocks by the corresponding entries of $\boldsymbol{\lambda}_t$. The results are subtracted from the affected entries of $\mathbf{b}_s$ and $\bar{\boldsymbol{\theta}}$, respectively. This computes both updates without assembling the full matrices $C_{t,s}$ and G_t. Once these updates are complete, H_t is no longer needed: the next reverse step rebuilds the Hessian at the preceding frame's converged state. The full trajectory calculation thus requires only one frame Hessian at a time, rather than the assembled matrix $\mathcal{J}$. This reverse pass, illustrated in Figure 8, yields the final parameter gradient after frame 1 has been processed. We next derive the inertia and friction contributions to $C_{t,s}$.

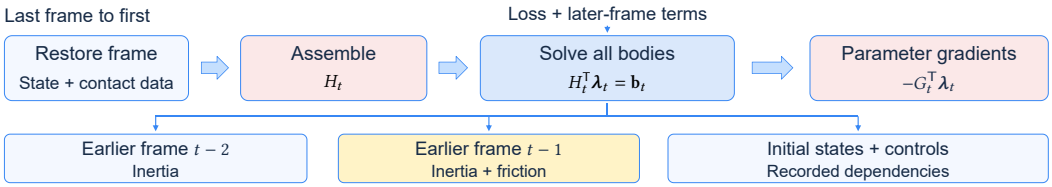


Figure 8: One reverse-frame step, repeated from T to 1. The saved state determines H_t, and the accumulated right-hand side $\mathbf{b}_t$ contains the loss gradient and contributions from later frames. Solving $H_t^\mathsf{T}\boldsymbol{\lambda}_t = \mathbf{b}_t$ supplies the parameter and earlier-state updates in Equations (53) and (54). For the illustrated BDF1 predictor, inertia contributes to $t-1$ and $t-2$, while friction contributes to $t-1$.

B.4 Inertia and Initial-State Dependencies

The history update in Equation (53) requires the derivative of $\mathbf{g}_t$ with respect to each earlier state. For inertia, $\mathbf{g}_t^{\text{in}} = M(\mathbf{y}_t - \widehat{\mathbf{y}}_t)$. Substituting the BDF1 predictor from Equation (16) and differentiating with respect to $\mathbf{y}_{t-1}$ and $\mathbf{y}_{t-2}$ gives

$$C_{t,t-1}^{\text{in}} = -2M, \qquad C_{t,t-2}^{\text{in}} = M. \tag{55}$$

Their contributions to Equation (53) are

$$\begin{aligned} \mathbf{b}_{t-1} &\mathrel{+}= 2M\boldsymbol{\lambda}_t, \quad t \geq 2, \\ \mathbf{b}_{t-2} &\mathrel{-}= M\boldsymbol{\lambda}_t, \quad t \geq 3. \end{aligned} \tag{56}$$

The same mass products apply to both representations: M contains the FEM vertex-mass blocks and the 12×12 mass block of each affine body.

The initial configuration and velocity are inputs to the simulation, not additional solved frames. Their contributions therefore enter the parameter update in Equation (54), rather than a right-hand-side vector $\mathbf{b}_0$. The first-step predictor is $\widehat{\mathbf{y}}_1 = \mathbf{y}_0 + h\dot{\mathbf{y}}_0 + h^2\mathbf{a}_1$. Substituting it into $\mathbf{g}_1^{\text{in}} = M(\mathbf{y}_1 - \widehat{\mathbf{y}}_1)$ gives partial derivatives $-M$ and $-hM$ with respect to $\mathbf{y}_0$ and $\dot{\mathbf{y}}_0$, respectively. At each frame, these partial derivatives occupy the initial-state parameter columns of G_t; other parameter columns are unchanged. We evaluate these derivatives at the converged state of frame t and accumulate their contributions to the initial-state gradients using Equation (54).

To approximate quasi-static behavior, we set the velocity used in the backward Euler predictor to zero at every step. The predictor becomes $\widehat{\mathbf{y}}_t = \mathbf{y}_{t-1} + h^2\mathbf{a}_t$, while the mass-weighted quadratic term in Equation (15) is retained. Differentiating its energy gradient with respect to $\mathbf{y}_{t-1}$ contributes $-M$ to $C_{t,t-1}$. Substituting this contribution into Equation (53) gives

$$\mathbf{b}_{t-1} \mathrel{+}= M\boldsymbol{\lambda}_t, \qquad t \geq 2. \tag{57}$$

B.5 Friction between Frames

Friction also adds a history derivative because its energy depends on the current and previous configurations, as described in Section A.2. DiffIPC includes this dependence for finite elements (Huang et al., 2024c); here we map the same type of derivative to both FEM and affine DoFs. For contact c, let $\mathbf{g}_{t,c}^{\text{fr},x}$ be the friction energy gradient with respect to the current world-space point positions $\mathbf{x}_t^c$. The maps $K_{t,c}$ and $K_{t-1,c}$ relate the same contact points to the simulation DoFs at the two frames. With rest geometry fixed, these maps do not depend on the solved DoFs. Define the point-space history derivative by

$$C_{t,c}^{\text{fr},x} = \frac{\partial \mathbf{g}_{t,c}^{\text{fr},x}}{\partial \mathbf{x}_{t-1}^c} \in \mathbb{R}^{3n_c \times 3n_c}. \tag{58}$$

Differentiating $\mathbf{g}_{t,c}^{\text{fr},y} = K_{t,c}^\mathsf{T}\mathbf{g}_{t,c}^{\text{fr},x}$ with respect to $\mathbf{y}_{t-1}$ therefore gives

$$\frac{\partial \mathbf{g}_{t,c}^{\text{fr},y}}{\partial \mathbf{y}_{t-1}} = K_{t,c}^\mathsf{T}\frac{\partial \mathbf{g}_{t,c}^{\text{fr},x}}{\partial \mathbf{x}_{t-1}^c}K_{t-1,c}. \tag{59}$$

This is the contact's contribution to $C_{t,t-1}$. For $t \geq 2$, substituting its transpose into Equation (53) gives

$$\mathbf{b}_{t-1} \mathrel{-}= K_{t-1,c}^{\mathsf{T}} \left(\frac{\partial \mathbf{g}_{t,c}^{\mathrm{fr},x}}{\partial \mathbf{x}_{t-1}^{c}} \right)^{\mathsf{T}} K_{t,c} \boldsymbol{\lambda}_t. \tag{60}$$

We first map $\boldsymbol{\lambda}_t$ to the contact points, apply the transposed point-space derivative, and map the result to the previous frame's DoFs. A contact can contain points from both representations; each point retains its FEM or affine map.

The relative tangential displacement is computed from the corresponding vertex displacements using closest-point interpolation weights and a tangent basis. These weights, the tangent basis, and the normal force are computed from the converged configuration of the previous frame. They remain fixed while solving the current frame. When differentiating with respect to the previous-frame positions, we also include the derivatives of these quantities. For ground friction, the normal is the prescribed $+Y$ direction and the tangent plane is X/Z.

C Physical-parameter derivatives

This appendix expands Sections 4.2 and 4.3. We give the mass and rest-shape derivatives for affine bodies, local FEM and cloth derivatives, friction-coefficient derivatives, initial-state terms, and joint controls. We evaluate parameter partial derivatives with the simulated DoFs at their converged values. Their transposed products with $\boldsymbol{\lambda}_t$ contribute to the loss gradient in Equation (51).

C.1 Rest-dependent mass and inertia

For affine body b, let Ω_b be its rest domain and ρ_b its constant density. For a material point with rest position $\bar{\mathbf{x}}$, the map $J(\bar{\mathbf{x}})$ in Equation (11) gives its world-space position as $\mathbf{x}_t = J(\bar{\mathbf{x}})\mathbf{q}_{t,b}$. With rest geometry fixed, its velocity is $\dot{\mathbf{x}}_t = J(\bar{\mathbf{x}})\dot{\mathbf{q}}_{t,b}$. The corresponding mass block is

$$M_b = \int_{\Omega_b} \rho_b J(\bar{\mathbf{x}})^{\mathsf{T}} J(\bar{\mathbf{x}}) \, \mathrm{d}V. \tag{61}$$

To compute M_b and its rest-shape derivatives, we evaluate the volume integral in Equation (61) as a sum of contributions from the triangles of the closed surface mesh. Each contribution is written in terms of the triangle's rest vertex positions. We differentiate these expressions with respect to those positions to obtain the mass-matrix derivatives, without constructing an interior tetrahedral mesh.

Mass from the closed surface. Expanding $J(\bar{\mathbf{x}})^{\mathsf{T}} J(\bar{\mathbf{x}})$ in Equation (61) gives terms that are constant, linear, or quadratic in the rest position $\bar{\mathbf{x}}$. We compute the mass matrix M_b using the following three integrals over the body's rest volume:

$$\begin{aligned} m_b &= \int_{\Omega_b} \rho_b \, \mathrm{d}V, \\ \mathbf{m}_b^{(1)} &= \int_{\Omega_b} \rho_b \bar{\mathbf{x}} \, \mathrm{d}V, \\ Q_b^{(2)} &= \int_{\Omega_b} \rho_b \bar{\mathbf{x}} \bar{\mathbf{x}}^{\mathsf{T}} \, \mathrm{d}V. \end{aligned} \tag{62}$$

Here m_b is the total mass. The first mass moment $\mathbf{m}_b^{(1)}$ integrates each rest-position component weighted by density. The second mass moment $Q_b^{(2)}$ integrates products of two rest-position components. With density fixed, these quantities depend on the body's rest geometry.

We compute the volume integrals as sums of signed contributions from the triangles bounding the rest domain. For an outward-oriented triangle f with rest vertices $\bar{\mathbf{x}}_{f,1}$, $\bar{\mathbf{x}}_{f,2}$, and $\bar{\mathbf{x}}_{f,3}$, the three vertices and the origin define a signed tetrahedron. Its volume and mass-weighted integrals can be calculated directly from the three vertex positions. With $\mathbf{s}_f = \bar{\mathbf{x}}_{f,1} + \bar{\mathbf{x}}_{f,2} + \bar{\mathbf{x}}_{f,3}$, the signed

contributions are

$$\begin{aligned} V_f &= \tfrac{1}{6}\bar{\mathbf{x}}_{f,1}^{\mathsf{T}}(\bar{\mathbf{x}}_{f,2}\times\bar{\mathbf{x}}_{f,3}),\\ m_f &= \rho_b V_f, \qquad \mathbf{m}_f^{(1)} = \tfrac{m_f}{4}\mathbf{s}_f,\\ Q_f^{(2)} &= \tfrac{m_f}{20}\left(\sum_{j=1}^{3}\bar{\mathbf{x}}_{f,j}\bar{\mathbf{x}}_{f,j}^{\mathsf{T}} + \mathbf{s}_f\mathbf{s}_f^{\mathsf{T}}\right). \end{aligned} \tag{63}$$

Summing m_f, $\mathbf{m}_f^{(1)}$, and $Q_f^{(2)}$ over all surface triangles gives m_b, $\mathbf{m}_b^{(1)}$, and $Q_b^{(2)}$, respectively. The signs are retained in these sums. This evaluates the volume integrals from the surface vertices without constructing an interior tetrahedral mesh.

Using these quantities and the coordinate order in Equation (9), the affine-body mass matrix is

$$M_b = \begin{bmatrix} m_b I_3 & \mathbf{e}_1(\mathbf{m}_b^{(1)})^{\mathsf{T}} & \mathbf{e}_2(\mathbf{m}_b^{(1)})^{\mathsf{T}} & \mathbf{e}_3(\mathbf{m}_b^{(1)})^{\mathsf{T}} \\ \mathbf{m}_b^{(1)}\mathbf{e}_1^{\mathsf{T}} & Q_b^{(2)} & 0 & 0 \\ \mathbf{m}_b^{(1)}\mathbf{e}_2^{\mathsf{T}} & 0 & Q_b^{(2)} & 0 \\ \mathbf{m}_b^{(1)}\mathbf{e}_3^{\mathsf{T}} & 0 & 0 & Q_b^{(2)} \end{bmatrix}. \tag{64}$$

The total mass determines the translation block. The first mass moment determines the blocks between translation and the affine transformation, while the second mass moment determines the blocks for the affine transformation.

Each contribution in Equation (63) is an explicit function of the triangle's rest vertex positions. With density and mesh connectivity fixed, we differentiate these expressions and sum the results over the surface triangles to obtain $\mathrm{d}m_b$, $\mathrm{d}\mathbf{m}_b^{(1)}$, and $\mathrm{d}Q_b^{(2)}$. Substituting them into Equation (64) gives $\mathrm{d}M_b$. When the inverse mass matrix is needed, its derivative is $\mathrm{d}(M_b^{-1}) = -M_b^{-1}(\mathrm{d}M_b)M_b^{-1}$. Shape updates preserve the closed, consistently oriented, nondegenerate surface and the validity of its mass distribution; invalid candidates are rejected before updating the simulation geometry.

C.2 Rest-shape gradients

Changing an affine body's rest shape affects its mass and inertia, affine shape energy, collision and friction terms. We compute their partial derivatives with respect to the surface-vertex rest positions $\bar{\mathbf{x}}_i$. During this differentiation, the FEM vertex positions and the affine translations $\mathbf{p}_{t,b}$ and matrices $A_{t,b}$ at each solved frame are treated as constants, using the values obtained from the converged forward solves. The affine surface can still change: a rest-position variation $\mathrm{d}\bar{\mathbf{x}}_i$ changes its world-space position by $\mathrm{d}\mathbf{x}_{t,i} = A_{t,b}\mathrm{d}\bar{\mathbf{x}}_i$. We denote the resulting first-order changes in the mass matrix, contact point map, and energy gradient by $\mathrm{d}M_b$, $\mathrm{d}K_{t,c}$, and $\mathrm{d}\mathbf{g}_t$, respectively. The partial derivatives of $\mathbf{g}_t$ with respect to the selected rest positions form the corresponding columns of G_t. A loss evaluated on the affine surface can also change directly with these rest positions; this contribution is included in $\boldsymbol{\ell}_\theta$.

Mass, inertia, and shape energy. For body b at frame t, the inertial energy gradient is $\mathbf{g}_{t,b}^{\mathrm{in}} = M_b(\mathbf{q}_{t,b} - \widehat{\mathbf{q}}_{t,b})$. The product rule gives the corresponding first-order change:

$$\mathrm{d}\mathbf{g}_{t,b}^{\mathrm{in}} = (\mathrm{d}M_b)(\mathbf{q}_{t,b} - \widehat{\mathbf{q}}_{t,b}) - M_b\,\mathrm{d}\widehat{\mathbf{q}}_{t,b}. \tag{65}$$

The first term includes the mass derivative derived above. The second includes any rest-shape dependence in the inputs to the predictor. The affine shape incremental potential is $h^2\kappa_{\mathrm{A}}V_b\psi_{\mathrm{A}}(\mathbf{q}_{t,b})$, where κ_{A} is the affine stiffness coefficient, V_b is the rest volume, and ψ_{A} is the near-rigidity penalty. Let $\mathbf{g}_{t,b}^{\mathrm{shape}}$ be its gradient with respect to $\mathbf{q}_{t,b}$. With κ_{A} and $\mathbf{q}_{t,b}$ fixed, the rest-shape derivative of the combined inertial and shape contributions is

$$\begin{aligned} \mathrm{d}(\mathbf{g}_{t,b}^{\mathrm{in}} + \mathbf{g}_{t,b}^{\mathrm{shape}}) = \mathrm{d}\mathbf{g}_{t,b}^{\mathrm{in}} \\ + h^2\kappa_{\mathrm{A}}(\mathrm{d}V_b)\nabla_{\mathbf{q}_{t,b}}\psi_{\mathrm{A}}. \end{aligned} \tag{66}$$

A rest-shape change affects not only the surface positions but also the mass matrix M_b and the volume factor V_b in the affine shape energy.

Collision-barrier derivatives and the point map. For a surface vertex i on body b, we vary its rest position $\bar{\mathbf{x}}_i$ while keeping the body's translation $\mathbf{p}_{t,b}$ and affine transformation $A_{t,b}$ unchanged. Equation (10) gives the resulting world-space position change as $\mathrm{d}\mathbf{x}_{t,i} = A_{t,b}\,\mathrm{d}\bar{\mathbf{x}}_i$. This variation also changes J_i and the corresponding entries of the contact map $K_{t,c}$. For contact c, let $\mathrm{d}\mathbf{x}_t^c$ collect the changes in its world-space point positions. Applying the product rule to the collision energy gradient $\mathbf{g}_{t,c}^{\mathrm{ct},y} = K_{t,c}^{\mathsf{T}}\mathbf{g}_{t,c}^{\mathrm{ct},x}$ gives its first-order change:

$$\begin{aligned}\mathrm{d}\mathbf{g}_{t,c}^{\mathrm{ct},y} = (\mathrm{d}K_{t,c})^{\mathsf{T}}\mathbf{g}_{t,c}^{\mathrm{ct},x} \\ + K_{t,c}^{\mathsf{T}}H_{t,c}^{\mathrm{ct},x}\mathrm{d}\mathbf{x}_t^c.\end{aligned} \tag{67}$$

Here ct denotes the collision-barrier term, and $H_{t,c}^{\mathrm{ct},x}$ is its Hessian with respect to the world-space point positions $\mathbf{x}_t^c$. The first term differentiates the map $K_{t,c}$. The second differentiates the world-space collision energy gradient as the contact-point positions change.

For a scalar rest-position component θ_k, we differentiate $\mathbf{g}_{t,c}^{\mathrm{ct},y} = K_{t,c}^{\mathsf{T}}\mathbf{g}_{t,c}^{\mathrm{ct},x}$ with respect to θ_k, keeping the simulation DoFs and other parameters unchanged. The product and chain rules give

$$\begin{aligned}\frac{\partial \mathbf{g}_{t,c}^{\mathrm{ct},y}}{\partial \theta_k} = \left(\frac{\partial K_{t,c}}{\partial \theta_k}\right)^{\mathsf{T}}\mathbf{g}_{t,c}^{\mathrm{ct},x} \\ + K_{t,c}^{\mathsf{T}}H_{t,c}^{\mathrm{ct},x}\frac{\partial \mathbf{x}_t^c}{\partial \theta_k}.\end{aligned} \tag{68}$$

$\partial \mathbf{g}_{t,c}^{\mathrm{ct},y}/\partial\theta_k$ is contact c's collision contribution to column k of G_t.

Rest shape under frictional contact. The friction energy at frame t depends on the surface-point positions at both frames t and $t-1$. For vertex i on affine body b, these two positions are computed from the same rest position $\bar{\mathbf{x}}_i$ using Equation (10). We vary $\bar{\mathbf{x}}_i$ while keeping the translation and affine transformation fixed at both frames. These are the converged values for solved frames and the prescribed initial values at frame 0. The friction coefficient and smoothing setting also remain fixed. The resulting world-space position changes are

$$\mathrm{d}\mathbf{x}_{t,i} = A_{t,b}\,\mathrm{d}\bar{\mathbf{x}}_i, \qquad \mathrm{d}\mathbf{x}_{t-1,i} = A_{t-1,b}\,\mathrm{d}\bar{\mathbf{x}}_i.$$

Both position changes enter the relative tangential displacement. The previous-frame change can also affect the closest-point interpolation weights, tangent basis, and normal force. We include all these dependencies in $\mathrm{d}\mathbf{g}_{t,c}^{\mathrm{fr},x}$, the first-order change in the world-space friction energy gradient. With $H_{t,c}^{\mathrm{fr},x} = \partial \mathbf{g}_{t,c}^{\mathrm{fr},x}/\partial \mathbf{x}_t^c$ and $C_{t,c}^{\mathrm{fr},x} = \partial \mathbf{g}_{t,c}^{\mathrm{fr},x}/\partial \mathbf{x}_{t-1}^c$, the point-space gradient change is

$$\mathrm{d}\mathbf{g}_{t,c}^{\mathrm{fr},x} = H_{t,c}^{\mathrm{fr},x}\,\mathrm{d}\mathbf{x}_t^c + C_{t,c}^{\mathrm{fr},x}\,\mathrm{d}\mathbf{x}_{t-1}^c. \tag{69}$$

The rest-shape variation also changes the map $K_{t,c}$. Applying the product rule to $\mathbf{g}_{t,c}^{\mathrm{fr},y} = K_{t,c}^{\mathsf{T}}\mathbf{g}_{t,c}^{\mathrm{fr},x}$ gives

$$\begin{aligned}\mathrm{d}\mathbf{g}_{t,c}^{\mathrm{fr},y} = (\mathrm{d}K_{t,c})^{\mathsf{T}}\mathbf{g}_{t,c}^{\mathrm{fr},x} \\ + K_{t,c}^{\mathsf{T}}\mathrm{d}\mathbf{g}_{t,c}^{\mathrm{fr},x}.\end{aligned} \tag{70}$$

The first term differentiates $K_{t,c}$; the second maps the world-space gradient change to the simulation DoFs. Contact-pair identities and closest-feature choices are fixed for this derivative evaluation. Point positions and the continuous quantities computed from them retain their parameter dependence.

Direct dependence of a surface loss. A loss evaluated on an affine surface also depends on rest geometry through $\mathbf{x}_{t,i} = J_i\mathbf{q}_{t,b}$. For a loss on surface positions at one frame t, holding the simulation DoFs fixed gives the direct contribution

$$\nabla_{\bar{\mathbf{x}}_i} L|_{\mathbf{y}_t \text{ fixed}} = A_{t,b}^{\mathsf{T}}\nabla_{\mathbf{x}_{t,i}} L. \tag{71}$$

For losses on several frames, these direct contributions are summed over the corresponding frames and included in $\boldsymbol{\ell}_\theta$. They are added to the contributions through the frame equations in Equation (51). If the loss is evaluated only on another object's state and has no explicit dependence on this rest position, its direct derivative for this parameter is zero. Its total derivative can still be nonzero because the shape change affects the coupled motion. After summing the frame-equation and direct surface-loss contributions, the chain rule maps the resulting rest-vertex gradient to the shape parameters.

C.3 FINITE-ELEMENT AND CLOTH PARAMETER DERIVATIVES

For a tetrahedral element e, let D_e^{rest} and D_e^{cur} contain its rest and current edge vectors, V_e its rest volume, and $\mathbf{x}_e$ its stacked vertex positions. The deformation gradient is $F_e = D_e^{\mathrm{cur}}(D_e^{\mathrm{rest}})^{-1}$. Let R_e denote the Jacobian of the vectorized deformation gradient with respect to the current element-vertex positions. The element's contribution to the step energy gradient is

$$\mathbf{g}_e = h^2 V_e R_e^{\mathsf{T}} \operatorname{vec}(P_e), \qquad R_e = \frac{\partial \operatorname{vec}(F_e)}{\partial \mathbf{x}_e}. \tag{72}$$

Here P_e is the first Piola stress and vec stacks the entries of F_e into a vector. For a tetrahedron, $R_e \in \mathbb{R}^{9\times 12}$ maps vertex-coordinate changes to deformation-gradient changes. The stable Neo-Hookean implementation stores two coefficients, θ_{len} and θ_{vol}, whose stress derivatives are

$$\begin{aligned} \frac{\partial P_e}{\partial \theta_{\mathrm{len}}} &= F_e - \operatorname{cof}(F_e), \\ \frac{\partial P_e}{\partial \theta_{\mathrm{vol}}} &= (\det F_e - 1)\operatorname{cof}(F_e). \end{aligned} \tag{73}$$

Substituting each stress derivative for P_e in Equation (72) gives the corresponding material-parameter derivative of $\mathbf{g}_e$. The API names `mu` and `lambda` address these stored coefficients, respectively, not an additional conversion from Young's modulus and Poisson's ratio. The same coefficient names are used by the task map in Equation (103).

At fixed current vertex positions, a rest-shape variation gives

$$\begin{aligned} \mathrm{d}\left[(D_e^{\mathrm{rest}})^{-1}\right] &= -(D_e^{\mathrm{rest}})^{-1}(\mathrm{d}D_e^{\mathrm{rest}})(D_e^{\mathrm{rest}})^{-1}, \\ \mathrm{d}F_e &= D_e^{\mathrm{cur}}\,\mathrm{d}\left[(D_e^{\mathrm{rest}})^{-1}\right], \end{aligned} \tag{74}$$

The full rest-shape derivative also includes changes in V_e, R_e, the stress, and the lumped mass. For a tetrahedral mesh, $m_i = \sum_{e\ni i} \rho_e V_e/4$. Cloth uses the corresponding 3×2 deformation gradient, area, thickness-weighted mass, and stretch and shear terms. For a bending hinge e, write

$$E_e^{\mathrm{bend}} = \tfrac{1}{2}\kappa_e^{\mathrm{bend}} \mathbf{x}_e^{\mathsf{T}} Q_e^{\mathrm{bend}}(\bar{\mathbf{x}}_e)\mathbf{x}_e, \tag{75}$$

where κ_e^{bend} is the bending coefficient and Q_e^{bend} the rest-dependent matrix for the hinge's four vertices. The partial derivative of the hinge's incremental energy gradient with respect to κ_e^{bend} is $h^2 Q_e^{\mathrm{bend}}\mathbf{x}_e$. Its rest derivative also differentiates Q_e^{bend}. The rest-dependent arrays are rebuilt when the shape is updated.

C.4 FRICTION-COEFFICIENT DERIVATIVES

At the converged frame state, the friction term contributes to the accumulated loss gradient for μ as

$$\bar{\mu} \mathrel{-}= \left[f_n W^{\mathsf{T}} U\, \nabla_{\boldsymbol{\xi}} \varphi(\|\boldsymbol{\xi}\|^2)\right]^{\mathsf{T}} K_{t,c}\boldsymbol{\lambda}_t. \tag{76}$$

Contributions are summed over contacts and frames. No further h^2 factor is applied.

C.5 INITIAL-STATE GRADIENTS

Following Section B.4, we include the selected components of an affine body's initial configuration $\mathbf{q}_{0,b}$ and initial velocity $\dot{\mathbf{q}}_{0,b}$ in $\boldsymbol{\theta}$. We now give their contributions to the parameter derivatives G_t and the gradient update in Equation (54). When computing these partial derivatives, only the selected initial values vary. The rest geometry and prescribed accelerations remain fixed, and $\mathbf{y}_1, \ldots, \mathbf{y}_T$ retain their converged forward values.

The first-frame inertial derivatives in Section B.4 use the mass block M_b for affine body b:

$$\frac{\partial \mathbf{g}_{1,b}^{\mathrm{in}}}{\partial \mathbf{q}_{0,b}} = -M_b, \qquad \frac{\partial \mathbf{g}_{1,b}^{\mathrm{in}}}{\partial \dot{\mathbf{q}}_{0,b}} = -hM_b. \tag{77}$$

The initial configuration also appears in the second-frame BDF1 predictor, $\widehat{\mathbf{q}}_{2,b} = 2\mathbf{q}_{1,b} - \mathbf{q}_{0,b} + h^2\mathbf{a}_{2,b}$. Differentiating $\mathbf{g}^{\text{in}}_{2,b} = M_b(\mathbf{q}_{2,b} - \widehat{\mathbf{q}}_{2,b})$ with respect to $\mathbf{q}_{0,b}$ gives

$$\frac{\partial \mathbf{g}^{\text{in}}_{2,b}}{\partial \mathbf{q}_{0,b}} = M_b. \tag{78}$$

The first-frame derivatives contribute to G_1, and the second-frame derivative contributes to G_2. Let $\boldsymbol{\lambda}_{t,b}$ contain the twelve entries of the coupled adjoint $\boldsymbol{\lambda}_t$ corresponding to body b. Substituting these derivatives into Equation (54) gives the inertial contributions to the accumulated initial-state gradients:

$$\begin{aligned} \bar{\mathbf{q}}_{0,b} &\mathrel{+}= M_b\boldsymbol{\lambda}_{1,b} - M_b\boldsymbol{\lambda}_{2,b}, \\ \bar{\dot{\mathbf{q}}}_{0,b} &\mathrel{+}= hM_b\boldsymbol{\lambda}_{1,b}. \end{aligned} \tag{79}$$

For a one-frame trajectory, the term containing $\boldsymbol{\lambda}_{2,b}$ is omitted.

Friction provides an additional initial-configuration derivative. At the first frame, its previous-frame positions are the initial surface positions $\mathbf{x}_{0,i} = J_i\mathbf{q}_{0,b}$. Changing $\mathbf{q}_{0,b}$ changes these positions and the interpolation weights, tangent basis, and normal force computed from them. We compute this derivative using Equation (59) with $t = 1$. Since $\mathbf{q}_{0,b}$ is an input parameter, this derivative contributes to G_1 and the parameter-gradient update, rather than an earlier-frame vector $\mathbf{b}_0$.

For the velocity-reset updates used to approximate quasi-static behavior in Section B.4, the first-frame predictor is $\widehat{\mathbf{q}}_{1,b} = \mathbf{q}_{0,b} + h^2\mathbf{a}_{1,b}$. The resulting inertial energy gradient has partial derivatives $-M_b$ with respect to $\mathbf{q}_{0,b}$ and zero with respect to $\dot{\mathbf{q}}_{0,b}$. Subsequent predictors use the previous solved configuration without an explicit $\mathbf{q}_{0,b}$ term. The inertial contribution to $\bar{\mathbf{q}}_{0,b}$ is therefore $M_b\boldsymbol{\lambda}_{1,b}$ in this mode; the first-frame friction contribution is included as described above.

C.6 Joint targets and efforts

We optimize the targets and conservative efforts of revolute and prismatic joints while keeping their reference geometry and driving stiffness fixed. These controls enter the joint-actuation energies; the geometric joint constraints remain unchanged. When joint j uses a scalar control $u_{j,t}$ directly at frame t, let $\mathbf{g}^{\text{joint}}_{t,j}$ denote its contribution to the frame energy gradient. Equation (54) gives the control-gradient contribution

$$\bar{u}_{j,t} \mathrel{-}= \left(\frac{\partial \mathbf{g}^{\text{joint}}_{t,j}}{\partial u_{j,t}}\right)^{\mathsf{T}} \boldsymbol{\lambda}_t. \tag{80}$$

This parameter derivative can be nonzero only in the entries corresponding to the joint's two bodies. The adjoint $\boldsymbol{\lambda}_t$ is computed from the coupled frame system, including collision and friction.

Reference geometry and coefficient scaling. Let a be the applied target, f the conservative effort, and k the effective driving stiffness. For the parent and child bodies, $\mathbf{q} = [\mathbf{q}_p^{\mathsf{T}}, \mathbf{q}_c^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^{24}$, and ∇ denotes differentiation with respect to $\mathbf{q}$. Rest geometry and joint reference directions are held fixed. For a material-space direction $\bar{\mathbf{a}}$, define

$$D(\bar{\mathbf{a}}) = [\, 0_{3\times 3}\ \mathbf{e}_1\bar{\mathbf{a}}^{\mathsf{T}}\ \mathbf{e}_2\bar{\mathbf{a}}^{\mathsf{T}}\ \mathbf{e}_3\bar{\mathbf{a}}^{\mathsf{T}}\,]. \tag{81}$$

This map gives $D(\bar{\mathbf{a}})\mathbf{q}_b = A_b\bar{\mathbf{a}}$ without a translation term. The material-space reference directions remain fixed. Their world-space transforms are evaluated from the current affine matrices without renormalization. The effective stiffness k is the specified driving strength multiplied by the joint-type strength ratio and the two-body mass sum stored at initialization. It remains fixed when differentiating targets and efforts. The displayed actuation energies are already contributions to the incremental potential: no additional h^2 factor is applied, and the effort coefficient f is used directly.

Prismatic coordinate, energy, and control derivatives. Let $\bar{\mathbf{x}}_p$ and $\bar{\mathbf{x}}_c$ be the material-space joint anchors, with $J_p = J(\bar{\mathbf{x}}_p)$ and $J_c = J(\bar{\mathbf{x}}_c)$. For the fixed child material axis $\bar{\mathbf{a}}$, define

$$\begin{aligned} \boldsymbol{\Delta} &= J_c\mathbf{q}_c - J_p\mathbf{q}_p, \qquad \mathbf{a} = D(\bar{\mathbf{a}})\mathbf{q}_c, \\ \xi &= \boldsymbol{\Delta}^{\mathsf{T}}\mathbf{a}, \\ \nabla\xi &= \begin{bmatrix} -J_p^{\mathsf{T}}\mathbf{a} \\ J_c^{\mathsf{T}}\mathbf{a} + D(\bar{\mathbf{a}})^{\mathsf{T}}\boldsymbol{\Delta} \end{bmatrix}. \end{aligned} \tag{82}$$

The second term in the child block differentiates the transformed axis with respect to the child's affine coordinates.

The actuation energy is added directly to the incremental potential. Its state gradient and the partial derivatives of that gradient with respect to the controls are

$$\begin{aligned}\Phi^{\mathrm{pri}} &= \tfrac{1}{2}k(\xi - a)^2 - f\xi,\\ \mathbf{g}^{\mathrm{pri}} &= \nabla\Phi^{\mathrm{pri}} = [k(\xi - a) - f]\nabla\xi,\\ \frac{\partial \mathbf{g}^{\mathrm{pri}}}{\partial a} &= -k\nabla\xi, \qquad \frac{\partial \mathbf{g}^{\mathrm{pri}}}{\partial f} = -\nabla\xi.\end{aligned} \tag{83}$$

Revolute coordinates, energy, and control derivatives. Each body has fixed material-space reference directions $\bar{\mathbf{n}}_b$ and $\bar{\mathbf{m}}_b$, with $b \in \{p, c\}$. Write $\mathbf{n}_b = A_b\bar{\mathbf{n}}_b$ and $\mathbf{m}_b = A_b\bar{\mathbf{m}}_b$. The relative-orientation quantities are

$$\begin{aligned}s_{\mathrm{r}} &= \tfrac{1}{2}(\mathbf{n}_c^{\mathsf{T}}\mathbf{m}_p - \mathbf{m}_c^{\mathsf{T}}\mathbf{n}_p),\\ c_{\mathrm{r}} &= \tfrac{1}{2}(\mathbf{n}_p^{\mathsf{T}}\mathbf{n}_c + \mathbf{m}_p^{\mathsf{T}}\mathbf{m}_c).\end{aligned} \tag{84}$$

Their gradients are

$$\begin{aligned}\nabla s_{\mathrm{r}} &= \tfrac{1}{2}\begin{bmatrix} D(\bar{\mathbf{m}}_p)^{\mathsf{T}}\mathbf{n}_c - D(\bar{\mathbf{n}}_p)^{\mathsf{T}}\mathbf{m}_c \\ D(\bar{\mathbf{n}}_c)^{\mathsf{T}}\mathbf{m}_p - D(\bar{\mathbf{m}}_c)^{\mathsf{T}}\mathbf{n}_p \end{bmatrix},\\ \nabla c_{\mathrm{r}} &= \tfrac{1}{2}\begin{bmatrix} D(\bar{\mathbf{n}}_p)^{\mathsf{T}}\mathbf{n}_c + D(\bar{\mathbf{m}}_p)^{\mathsf{T}}\mathbf{m}_c \\ D(\bar{\mathbf{n}}_c)^{\mathsf{T}}\mathbf{n}_p + D(\bar{\mathbf{m}}_c)^{\mathsf{T}}\mathbf{m}_p \end{bmatrix}.\end{aligned} \tag{85}$$

For target a, define

$$S_{\mathrm{r}} = s_{\mathrm{r}}\cos a - c_{\mathrm{r}}\sin a, \qquad C_{\mathrm{r}} = c_{\mathrm{r}}\cos a + s_{\mathrm{r}}\sin a. \tag{86}$$

The quantities s_{r} and c_{r} are not normalized: $s_{\mathrm{r}}^2 + c_{\mathrm{r}}^2$ need not equal one for general affine states. On a smooth, nondegenerate branch of $\alpha = \operatorname{atan2}(s_{\mathrm{r}}, c_{\mathrm{r}})$,

$$\begin{aligned}\nabla\alpha &= \frac{c_{\mathrm{r}}\nabla s_{\mathrm{r}} - s_{\mathrm{r}}\nabla c_{\mathrm{r}}}{s_{\mathrm{r}}^2 + c_{\mathrm{r}}^2},\\ \nabla S_{\mathrm{r}} &= \cos a\,\nabla s_{\mathrm{r}} - \sin a\,\nabla c_{\mathrm{r}},\\ \nabla C_{\mathrm{r}} &= \cos a\,\nabla c_{\mathrm{r}} + \sin a\,\nabla s_{\mathrm{r}}.\end{aligned} \tag{87}$$

The target energy uses S_{r} and C_{r} rather than a quadratic penalty on $\alpha - a$:

$$\begin{aligned}\Phi^{\mathrm{rev}} &= \tfrac{1}{2}k\big[S_{\mathrm{r}}^2 + w_c(1 - C_{\mathrm{r}})^2\big] - f\alpha,\\ \mathbf{g}^{\mathrm{rev}} &= \nabla\Phi^{\mathrm{rev}}\\ &= k\big[S_{\mathrm{r}}\nabla S_{\mathrm{r}} - w_c(1 - C_{\mathrm{r}})\nabla C_{\mathrm{r}}\big] - f\nabla\alpha,\end{aligned} \tag{88}$$

where $w_c = 0.02$. The partial derivatives with respect to the applied target and effort are

$$\begin{aligned}\frac{\partial \mathbf{g}^{\mathrm{rev}}}{\partial a} &= k\big[(-C_{\mathrm{r}} - w_c(1 - C_{\mathrm{r}}))\nabla S_{\mathrm{r}}\\ &\qquad + (w_c - 1)S_{\mathrm{r}}\nabla C_{\mathrm{r}}\big],\\ \frac{\partial \mathbf{g}^{\mathrm{rev}}}{\partial f} &= -\nabla\alpha.\end{aligned} \tag{89}$$

The relations $\partial_a S_{\mathrm{r}} = -C_{\mathrm{r}}$, $\partial_a C_{\mathrm{r}} = S_{\mathrm{r}}$, $\partial_a\nabla S_{\mathrm{r}} = -\nabla C_{\mathrm{r}}$, and $\partial_a\nabla C_{\mathrm{r}} = \nabla S_{\mathrm{r}}$ give Equation (89). The implementation floors the denominator in $\nabla\alpha$ at 10^{-20}. This safeguard is not an exact derivative at a degenerate orientation. Angle wrapping and branch crossings are not assumed differentiable.

All control derivatives are evaluated at the converged frame states.

Joint blocks used in the linear systems. The joint-actuation blocks contributing to H_t are

$$\begin{aligned} H^{\mathrm{pri}}_{\mathrm{joint}} &= k\,\nabla\xi\,\nabla\xi^{\mathsf{T}}, \\ H^{\mathrm{rev}}_{\mathrm{joint}} &= k\big(\nabla S_{\mathrm{r}}\,\nabla S_{\mathrm{r}}^{\mathsf{T}} + w_c \nabla C_{\mathrm{r}}\,\nabla C_{\mathrm{r}}^{\mathsf{T}}\big). \end{aligned} \tag{90}$$

These 24×24 blocks are used in the numerical linear system and retain the parent–child coupling. The target and effort derivatives in Equations (83) and (89) differentiate the energy gradients directly. Fixed-body entries are excluded when combining the local parameter derivatives with the frame adjoint. The effort model is conservative; these formulas do not specify a separate non-conservative actuation model.

Target selection, limiting, and shared controls. A per-frame target or effort overrides a persistent value on that frame. A direct parameter-gradient contribution is accumulated only for the value actually applied. The applied target can differ from the command $u_{j,t}$. We limit its change relative to the joint coordinate $\xi_{j,t-1}$ in the previous simulated state:

$$\begin{aligned} \delta_{j,t} &= w_j(u_{j,t} - \xi_{j,t-1}), \\ \widetilde{u}_{j,t} &= \xi_{j,t-1} + \mathrm{clip}(\delta_{j,t}, -\Delta_j, \Delta_j). \end{aligned} \tag{91}$$

Here $\xi_{j,t-1}$ is the prismatic displacement or revolute angle, and Δ_j is the per-frame limit. The difference map is $w_j(\delta) = \delta$ for a prismatic joint and

$$w_j(\delta) = \delta - 2\pi\,\mathrm{round}(\delta/(2\pi))$$

for a revolute joint. We first use Equation (80) with $\widetilde{u}_{j,t}$ as the applied target to compute $\overline{\widetilde{u}}_{j,t}$. For the limiter branch used by the forward step, define

$$\gamma_{j,t} = \begin{cases} 1, & |\delta_{j,t}| < \Delta_j, \\ 0, & |\delta_{j,t}| \geq \Delta_j. \end{cases} \tag{92}$$

We update the command gradient and the previous-frame right-hand side as

$$\begin{aligned} \bar{u}_{j,t} \mathrel{+}&= \gamma_{j,t}\,\overline{\widetilde{u}}_{j,t}, \\ \mathbf{b}_{t-1} \mathrel{+}&= (1-\gamma_{j,t})\nabla_{\mathbf{y}_{t-1}}\xi_{j,t-1}\,\overline{\widetilde{u}}_{j,t}, \qquad t \geq 2. \end{aligned} \tag{93}$$

When the limit is active, the applied target depends on the previous joint coordinate rather than the command. At $t = 1$, this contribution is accumulated for the selected initial-state parameters instead of $\mathbf{b}_0$. These derivatives apply within a fixed limiting and angle branch. Equality with the limit uses the saturated convention, without a differentiability claim at that boundary. Finite-difference checks must keep the limiting and angular branches unchanged. Efforts do not use this target limiter. When one optimizer variable supplies a shared control to several joints or frames, we sum their corresponding gradient contributions.

D GPU IMPLEMENTATION AND INTERFACE DETAILS

Our forward simulator is built on StiffGIPC (Huang et al., 2025a). We inherit its global Hessian assembly for the coupled FEM–affine system and its 3×3 sparse block representation. For the backward pass, we use this assembly method to construct H_t from the element, body, collision, friction, and joint terms at each frame's converged state. We describe frame restoration, the sparse matrix products and PCG stopping rules, CUDA Graph execution, and parallel local-gradient accumulation in one place. The final part gives the PyTorch interface, parameter registry, and supported combinations.

D.1 RESTORING CONVERGED FRAME STATES

The derivatives in the frame-wise adjoint are evaluated at the converged states of the forward simulation. After each forward step, we save the FEM vertex positions and velocities, affine coordinates and velocities, and the collision, friction, and joint-control data used by that frame. During the reverse pass, we restore these values for frame t. The restored FEM positions directly specify the deformable geometry; for each affine body, Equation (10) recovers the world-space surface positions

from its translation and affine transformation. The time step, physical parameters, and DoF ordering remain those of the recorded simulation.

We also recover the earlier-frame data required by the frame equation. The BDF1 predictor uses the two previous configurations, while friction uses the previous configuration's interpolation weights, tangent basis, and normal force together with the current surface positions. Joint-target limits use the recorded control values and the previous joint configuration. We retain the recorded contact pairs and closest-feature choices when evaluating their collision and friction derivatives. The continuous dependence on the earlier positions is included as derived in Sections B.5 and C.6. With the current and earlier-frame data restored, we evaluate the local derivatives and assemble H_t without repeating the nonlinear forward solve. The Hessian is rebuilt for each reverse frame; only the current frame's matrix is retained.

D.2 GPU ADJOINT SOLVE

The trajectory equations in Equation (43) give the adjoint system in Equation (46). Following Section B.3, we solve this system in reverse frame order using $H_t^{\mathsf{T}}\boldsymbol{\lambda}_t = \mathbf{b}_t$ (Equation (52)), without assembling the trajectory matrix. For each frame, we use preconditioned conjugate gradients (PCG) with the assembled H_t and accumulated right-hand side $\mathbf{b}_t$. The matrix, right-hand side, and preconditioner remain fixed during this linear solve.

D.2.1 PCG ALGORITHM AND STOPPING RULES

For a fixed frame, the assembled matrix is symmetric, so the adjoint system is $H_t\boldsymbol{\lambda}_t = \mathbf{b}_t$. The matrix H_t, right-hand side $\mathbf{b}_t$, and preconditioner $\mathcal{P}_t$ remain unchanged during this calculation. Algorithm 2 computes $\boldsymbol{\lambda}_t$ from an initial estimate $\boldsymbol{\lambda}^{(0)}$. Here $\mathbf{r}$ is the linear residual, $\mathbf{w}$ is the preconditioned residual obtained by solving $\mathcal{P}_t\mathbf{w} = \mathbf{r}$, and $\mathbf{d}$ is the PCG search direction. The notation $\mathcal{P}_t^{-1}\mathbf{r}$ denotes application of the preconditioner, not formation of its inverse.

The algorithm first computes the residual and initializes the search direction. Each iteration multiplies $\mathbf{d}$ by H_t, updates the adjoint estimate and residual, applies the preconditioner, and forms the next search direction. We monitor the scalar $\chi = \mathbf{r}^{\mathsf{T}}\mathbf{w}$ against its initial magnitude χ_{ref}. The stopping test is $|\chi| \le \eta_{\mathrm{cg}}\chi_{\mathrm{ref}}$, a relative preconditioned residual-energy test, not a Euclidean residual-norm test. The default backward threshold η_{cg} is at most 10^{-14}.

Before accepting the adjoint, we recompute $\mathbf{r}_{\mathrm{true}} = \mathbf{b}_t - H_t\boldsymbol{\lambda}$ rather than relying only on the recursively updated residual. The recomputed residual must pass the configured final residual check. If it does not, the solver may restart from the current estimate, at most twice, provided that the recomputed residual improves on the previous directly computed residual according to the configured progress check, and the iteration budget has not been exhausted. A restart resets the search direction, but retains the original χ_{ref}, tolerance, and iteration count. The final residual and restart-progress checks use the same solver settings throughout the calculation.

Algorithm 2 PCG for one frame's coupled adjoint system

Input: H_t, $\mathbf{b}_t$, fixed preconditioner $\mathcal{P}_t$, initial estimate $\boldsymbol{\lambda}^{(0)}$, relative threshold η_{cg}, total iteration limit $K_{\max}$; configured final residual and restart-progress checks
Output: Adjoint $\boldsymbol{\lambda}_t$ satisfying the residual checks, or failure

1: $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}^{(0)}$, $k \leftarrow 0$, $j \leftarrow 0$ ▷ iteration and restart counts
2: $\mathbf{r} \leftarrow \mathbf{b}_t - H_t\boldsymbol{\lambda}$, $\mathbf{w} \leftarrow \mathcal{P}_t^{-1}\mathbf{r}$, $\chi \leftarrow \mathbf{r}^{\mathsf{T}}\mathbf{w}$
3: $\chi_{\mathrm{ref}} \leftarrow |\chi|$ ▷ keep this reference unchanged after restarts
4: $\mathbf{r}_{\mathrm{prev}} \leftarrow \mathbf{r}$ ▷ previous directly computed residual
5: **if** $\mathbf{r} = \mathbf{0}$ **then**
6: **return** $\boldsymbol{\lambda}_t \leftarrow \boldsymbol{\lambda}$
7: **end if**
8: **while** $k < K_{\max}$ **do**
9: $\mathbf{d} \leftarrow \mathbf{w}$ ▷ initial or restarted search direction
10: **repeat**
11: $\mathbf{v} \leftarrow H_t\mathbf{d}$, $\alpha \leftarrow \chi/(\mathbf{d}^{\mathsf{T}}\mathbf{v})$ ▷ matrix product and step length
12: $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \alpha\mathbf{d}$, $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{v}$ ▷ update estimate and residual
13: $\mathbf{w} \leftarrow \mathcal{P}_t^{-1}\mathbf{r}$, $\chi_{\mathrm{new}} \leftarrow \mathbf{r}^{\mathsf{T}}\mathbf{w}$
14: $\beta \leftarrow \chi_{\mathrm{new}}/\chi$, $\mathbf{d} \leftarrow \mathbf{w} + \beta\mathbf{d}$ ▷ next search direction
15: $\chi \leftarrow \chi_{\mathrm{new}}$, $k \leftarrow k + 1$
16: **until** $|\chi| \le \eta_{\mathrm{cg}}\chi_{\mathrm{ref}}$ or $k = K_{\max}$
17: $\mathbf{r}_{\mathrm{true}} \leftarrow \mathbf{b}_t - H_t\boldsymbol{\lambda}$ ▷ recompute, rather than update, the residual
18: **if** $\mathbf{r}_{\mathrm{true}}$ passes the configured final residual check **then**
19: **return** $\boldsymbol{\lambda}_t \leftarrow \boldsymbol{\lambda}$
20: **end if**
21: **if** $k = K_{\max}$ or $j = 2$ or the progress check on $(\mathbf{r}_{\mathrm{true}}, \mathbf{r}_{\mathrm{prev}})$ fails **then**
22: **return** failure
23: **end if**
24: $\mathbf{r}_{\mathrm{prev}} \leftarrow \mathbf{r}_{\mathrm{true}}$, $\mathbf{r} \leftarrow \mathbf{r}_{\mathrm{true}}$
25: $\mathbf{w} \leftarrow \mathcal{P}_t^{-1}\mathbf{r}$, $\chi \leftarrow \mathbf{r}^{\mathsf{T}}\mathbf{w}$, $j \leftarrow j + 1$ ▷ restart from the current estimate
26: **end while**
27: **return** failure

Failure stops the frame's backward calculation before its adjoint is used in parameter or earlier-frame updates. An explicitly specified iteration limit is not increased after failure. When no limit is supplied, the default backward budget is at least the number of scalar simulation DoFs. Run-specific settings can override these defaults. The next two parts describe the sparse products in the iteration loop and its execution with CUDA Graphs.

D.2.2 Symmetric sparse matrix products

The sparse matrix products use the inherited symmetric block format. For $\mathbf{w} = H_t\mathbf{v}$, with $\mathbf{w}$ initialized to zero, a stored off-diagonal block contributes

$$\begin{aligned} \mathbf{w}_r &\mathrel{+}= (H_t)_{rs}\mathbf{v}_s, \\ \mathbf{w}_s &\mathrel{+}= (H_t)_{rs}^{\mathsf{T}}\mathbf{v}_r, \qquad r < s. \end{aligned} \tag{94}$$

Here r, s index three-component blocks, and each diagonal block contributes once. PCG uses these products with the configured block or multilevel preconditioner.

D.2.3 CUDA Graph execution and reuse

Algorithm 2 separates the repeated PCG loop (lines 10–16) from direct residual verification (line 17) and a possible restart (line 25). Matrix products, preconditioning, vector updates, and dot-product reductions are parallel GPU operations. We capture initialization and the repeated iteration operations in a conditional CUDA Graph. A GPU kernel tests convergence and updates k, so the graph continues or exits without asking the CPU to check each iteration. On exit, the residual is recomputed and checked before the adjoint is accepted or the search direction is restarted. This changes how the operations are launched, not their order, precision, or stopping criteria. The original tolerance and total iteration budget remain unchanged after a restart.

Before each solve, we update the graph's inputs with the current matrix and vector addresses, dimensions, and solver settings. The graph can be reused while its launch configuration, preconditioner storage, and work buffers remain compatible; otherwise, it is rebuilt. Reusing the graph does not reuse the previous frame's Hessian values.

If graph capture is unavailable, a CPU-controlled loop executes the same kernels. A CUDA execution error is reported without retrying the calculation on partially updated solver data. Backward vectors and graph objects remain separate from those of the forward solver. These execution conventions do not change the PCG recurrence or the requested tolerance.

Figure 9 summarizes the restored-frame assembly, solve, and gradient accumulation.

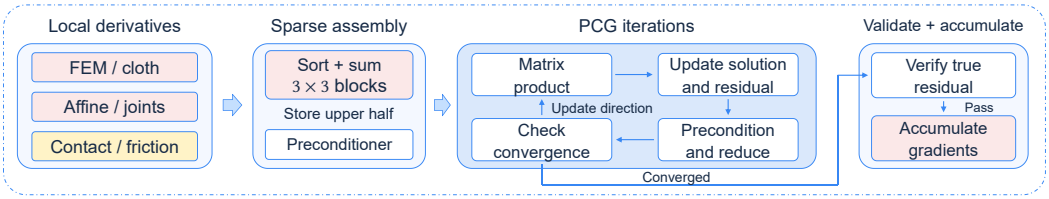


Figure 9: GPU backward computation for one converged frame. The inherited sparse assembly forms the coupled Hessian from local energy derivatives. A conditional CUDA graph repeats the PCG matrix products, preconditioning, vector updates, and reductions, with convergence checked on the GPU. After the adjoint solve, local parameter and history derivatives are evaluated and accumulated in parallel.

D.3 Parallel history and parameter accumulation

Equation (95) defines the local derivative records and their parameter-gradient products:

$$\boldsymbol{\gamma}_\ell = \frac{\partial \mathbf{g}_{t,r_\ell}^{[\ell]}}{\partial \theta_{k_\ell}}, \qquad \bar{\theta}_k \mathrel{-}= \sum_{\ell:k_\ell=k} \boldsymbol{\gamma}_\ell^{\mathsf{T}} \boldsymbol{\lambda}_{t,r_\ell}. \tag{95}$$

Here r_ℓ indexes a three-component energy-gradient block, k_ℓ indexes a scalar physical parameter, and the GPU sums products that contribute to the same parameter. A kernel can store the records or directly accumulate their products with the corresponding adjoint entries. Contributions sharing a global row and parameter column are summed; neither choice requires storing the complete matrix G_t.

History updates use the same approach. With $\boldsymbol{\lambda}_t$ fixed, each local term computes its contribution to an earlier-frame vector $\mathbf{b}_s$. Inertia uses the mass products in Section B.4, and friction uses the point-space product in Equation (60). Joint-target limits add the previous-state contributions in Equation (93). The local contributions are evaluated in parallel and summed at their destination DoFs without assembling the full history matrix $C_{t,s}$. After these updates, the reverse pass proceeds to frame $t-1$. Once all frames have been processed, $\bar{\boldsymbol{\theta}}$ is the physical parameter gradient returned to the interface described next.

D.4 A reusable interface for inverse problems

The GPU backward calculation takes the loss gradients $\boldsymbol{\ell}_t$ and $\boldsymbol{\ell}_\theta$ as inputs and returns $\bar{\boldsymbol{\theta}}$. Our PyTorch-based Python interface computes these inputs from user-defined outputs and losses, and converts the returned physical-parameter gradient into a gradient with respect to the optimizer variables. This separates task definition from the physical derivative implementation.

D.4.1 TASK DEFINITION

Let $\mathbf{z} \in \mathbb{R}^{n_z}$ contain the optimizer variables, ϕ map them to the selected physical parameters $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$, and $\mathcal{O}$ select or transform the simulation states used by a scalar loss function ℓ:

$$\begin{aligned} \boldsymbol{\theta} &= \phi(\mathbf{z}), \\ \mathbf{o} &= \mathcal{O}(\mathbf{y}_1, \ldots, \mathbf{y}_T, \boldsymbol{\theta}), \\ L &= \ell(\mathbf{o}, \boldsymbol{\theta}). \end{aligned} \tag{96}$$

The map ϕ can enforce positivity, share optimizer variables across physical parameters, or parameterize a rest shape, so n_z need not equal n_θ. The output map can select FEM positions, affine DoFs, or world-space affine surface points from one or more frames. Both maps and the loss are defined in Python.

D.4.2 CONNECTING THE LOSS TO THE BACKWARD CALCULATION

PyTorch differentiates the loss and output map to obtain

$$\boldsymbol{\ell}_o = \nabla_{\mathbf{o}} \ell, \qquad \boldsymbol{\ell}_t = \left(\frac{\partial \mathcal{O}}{\partial \mathbf{y}_t} \right)^{\mathsf{T}} \boldsymbol{\ell}_o. \tag{97}$$

For world-space affine points, this calculation uses J_i^{T} as in Equation (38); FEM positions use the identity map. A frame not used by $\mathcal{O}$ has $\boldsymbol{\ell}_t = \mathbf{0}$. The loss and output map can also depend explicitly on the physical parameters. With the simulated DoFs fixed, this gives

$$\boldsymbol{\ell}_\theta = \nabla_{\boldsymbol{\theta}} \ell|_{\mathbf{o} \text{ fixed}} + \left(\frac{\partial \mathcal{O}}{\partial \boldsymbol{\theta}} \right)^{\mathsf{T}} \boldsymbol{\ell}_o. \tag{98}$$

For example, affine rest-shape optimization includes the direct change in surface positions at fixed affine coordinates. The vectors $\boldsymbol{\ell}_t$ initialize the frame right-hand sides $\mathbf{b}_t$, and $\boldsymbol{\ell}_\theta$ initializes the parameter-gradient accumulation. The GPU adjoint then computes $\bar{\boldsymbol{\theta}}$, and PyTorch applies the derivative of ϕ to obtain

$$\begin{aligned} \bar{\boldsymbol{\theta}} &= \boldsymbol{\ell}_\theta - \sum_{t=1}^{T} G_t^{\mathsf{T}} \boldsymbol{\lambda}_t, \\ \nabla_{\mathbf{z}} L &= \left(\frac{\partial \phi}{\partial \mathbf{z}} \right)^{\mathsf{T}} \bar{\boldsymbol{\theta}}. \end{aligned} \tag{99}$$

D.4.3 PARAMETER SELECTION AND SUPPORTED COMBINATIONS

Before simulation, each selected parameter block is assigned a fixed position in $\boldsymbol{\theta}$. This ordering is retained when frames are restored, so accumulated derivatives refer to the same physical parameters throughout the backward pass. Supported parameter groups include material and friction coefficients, initial states, FEM and affine rest positions, and joint targets and efforts. Changing the parameter map, output selection, or loss reuses the same backward solver without task-specific backward code. The interface reuses implemented physical derivatives. A new physical law or an output not expressible from supported quantities requires the corresponding derivative implementation.

Table 2 identifies the implemented parameter families. Affine rest positions support all contact cases handled by the differentiable framework, including active friction. This does not imply that every parameter combination is supported. In particular, the current implementation does not support FEM rest-shape differentiation in a system containing affine DoFs. Static-equilibrium differentiation supports the specified pure-FEM and cloth-rest cases, but not volumetric rest parameters, active friction, initial velocity, or mixed FEM–affine states. Revolute and prismatic targets and conservative efforts are supported with fixed joint reference geometry and driving stiffness; other forward joint types do not automatically acquire control derivatives.

Supported augmented-Lagrangian state can be restored, but its targets, weights, and multipliers are not differentiated; active augmented-Lagrangian recovery with friction is unsupported. Ground derivatives use the prescribed plane. The Boolean settings that enable or disable collision between geometry pairs are not differentiated.

Table 2: Registered parameter names. These rows list parameter families, not a claim that every combination is implemented.

Physical group	Registry name	Meaning
Tetrahedral material	`mu`, `lambda`	Stored stable Neo-Hookean coefficients
Initial FEM motion	`v0`	Selected vertex velocities
FEM/cloth rest shape	`rest`	Selected rest coordinates
Cloth material	`tri_stretch`, `tri_shear`, `bend`	Stretch, shear, and bending coefficients
Friction	`friction`, `gd_friction`	Global self and ground friction coefficients
Initial affine motion	`abd_q0`, `abd_v0`	Affine coordinates and velocities
Affine rest shape	`abd_rest`	Selected rest coordinates, including collision and friction
Joint controls	`abd_joint_target_`*id*, `abd_joint_effort_`*id*	Revolute or prismatic target/effort; optional frame selection

D.4.4 Examples of task composition

Table 3 lists the parameter and output choices in the eleven tasks. Material fitting uses positive coefficient maps, friction optimization shares one coefficient between object–object and ground contact, Garment Hanger design maps two shoulder scales to rest coordinates, and dual-arm grasping shares one aperture across four joints. These tasks change the parameter map and objective while reusing the physical derivatives and GPU adjoint described above.

Table 3: How the same interface is used across the eleven tasks. The map, selected output, and loss change at the application level. Physical derivatives remain limited to the implemented parameter types and their supported combinations. Task checks can inspect more quantities than the optimized loss. Parentheses give figures illustrating tasks in each group.

Task group	Values selected by the application	Output used by the loss
Contact-driven motion (Fig. 10, Fig. 11)	Shared initial velocity	Other body's final center or a multi-frame trajectory
Material identification and friction optimization (Fig. 13, Fig. 14)	Positive material coefficients or a shared friction value	Loaded markers or the bunny's position at the specified time
Shape design (Fig. 17a, Fig. 12, Fig. 16)	FEM rest vertices, cloth rest scale, or two hanger shoulder scales	Stress, drape, or garment support
Placement (Fig. 15)	Horizontal components of initial affine states	Stack motion and the specified stability objective
Articulated control (Fig. 17b, Fig. 18, Fig. 19)	Joint schedule, shared target angle, or shared aperture	Plate motion, actual finger angles, or actual opening

E Experimental setup, tasks, and measurements

We evaluate DiffGIPC on eleven inverse problems involving motion, material identification, friction optimization, rest-shape design, placement, and articulated control. The experiments examine whether the same backward method supports these parameter and loss choices, and how its cost changes with object representation and problem size. We give the simulation settings, measurement scope, and visualization conventions first, followed by the individual tasks, comparisons with DiffIPC, representation studies, and complete recorded workloads.

E.1 Experimental setup and measurement scope

The hardware and floating-point precision are specified in Section 6. Table 4 specifies the simulation settings. Table 3 identifies the parameter and output choices, and Table 7 gives the measured workload for each task. A forward evaluation advances a simulation over its configured horizon; a backward evaluation computes the parameter gradient for that simulation. Neither count denotes an individual Newton iteration or a single frame's adjoint system.

E.1.1 Simulation settings and stopping conditions

Table 4 gives the eleven-task settings. The separate bunny and Garment Hanger representation studies use the model variations specified in Section E.6.

Table 4: Simulation settings for the eleven experiments. h is the time step, d_{rel} is the relative collision activation distance, and ϵ_N is the relative Newton tolerance. $\widehat{d}_f$ denotes the configured friction-smoothing value. E is the FEM or cloth stretching Young's modulus; κ_{A} is the affine stiffness.

#	Example	h	Frames	d_{rel}	κ	$\widehat{d}_f$	#DOF	Newton ϵ_N	Friction	E	κ_{A}
1	Stone placement (Fig. 15)	0.03	1000	10^{-3}	10^5	10^{-3}	84	10^{-3}	0.3	—	10^7
2	Bunny collision (Fig. 10)	0.05	40	10^{-3}	10^{10}	—	5598	3×10^{-4}	0	4×10^7	—
3	Tentacle matching (Fig. 11)	0.005	100	10^{-3}	10^7	10^{-3}	20688	10^{-2}	1	2×10^4	—
4	Garment Hanger (Fig. 17a)	$1/120$	600	10^{-3}	10^5	10^{-3}	51357	10^{-2}	0.05	5×10^4	10^8
5	Loaded Support (Fig. 12)	0.02	20	3×10^{-4}	10^6	—	22545	10^{-3}	0	10^8	—
6	Material fitting (Fig. 13)	0.1	10	10^{-3}	10^7	—	12567	10^{-2}	0	10^6	—
7	Friction optimization (Fig. 14)	0.02	100	10^{-3}	10^6	10^{-3}	2799	10^{-5}	0.5→0.1	10^5	—
8	Cloth rest scale (Fig. 16)	$1/24$	48	10^{-3}	10^5	—	102732	10^{-3}	0	10^3	10^8
9	Grasp angle (Fig. 17b)	0.02	120	10^{-3}	10^5	10^{-3}	195	5×10^{-3}	0.1	5×10^5	10^7
10	Cup catching (Fig. 18)	0.005	240	10^{-3}	10^5	10^{-3}	192	10^{-3}	2.0	—	10^7
11	Dual-arm grasp (Fig. 19)	0.01	300	10^{-3}	10^5	10^{-3}	12504	10^{-2}	0.4	300	10^9

Let L_{box} be the scene bounding-box diagonal. The configured relative collision distance d_{rel} gives the physical activation gap $d_{\mathrm{rel}}L_{\mathrm{box}}$. Its square is $\widehat{d}$ in the collision-barrier equation. The configured friction-smoothing value $\widehat{d}_f$ supplies $\epsilon_{\mathrm{fr}} = h\widehat{d}_f$. The affine shape coefficient κ_{A} is neither a Young's modulus nor the collision-barrier coefficient κ. For friction optimization, the coefficient is fixed within each simulation and updated between optimization iterations. The recorded Newton stopping rule tests the current world-space direction:

$$\|\delta\mathbf{x}\|_\infty < \epsilon_N h L_{\mathrm{box}}, \tag{100}$$

where ϵ_N and h are the relative tolerance and time step in Table 4.

E.1.2 Timing conventions

We report mean forward and backward times separately. Their sum is the cost of one mean forward evaluation plus one mean backward evaluation; accumulated evaluation time instead sums all recorded calls, including line-search trials and verification replays. Table 7 gives the eleven-task breakdown. *Linear*, *CCD*, *Hessian*, and *Search* denote linear systems, collision detection, Hessian assembly, and forward line search. *Other* is the independently measured total minus the named phases; it is not one kernel.

The Garment Hanger means combine thirty cutoff-limited optimization rollouts and two complete 600-frame checks. The dual-arm means combine thirteen 142-frame rollouts and two 300-frame checks. The forward-plus-backward totals in Tables 5 and 6 have a different meaning from the accumulated total: they add one mean forward and one mean backward call. No mean is converted to a fixed per-frame cost for these mixed horizons.

E.1.3 VISUALIZATION AND LOSS-PLOT CONVENTIONS

The stone panels show updates 0, 10, 50, and 78 of the measured differentiable-simulation run, each replayed for 1,000 frames. Its first 50 updates use centroid tolerance 0.01 in the loss, and later updates use 0.15. The Loaded Support visualization uses the power-2 stress run with arm-normal cuts and a shared RMS first-Piola stress color scale. The cube panels show frame 19, and the friction panels show frame 100.

The grasp-angle images compare commands 0.39 and 0.447843 rad for an objective targeting four joint angles at 0.45 rad. The cup images show controls after 40 updates. The dual-arm images show the same recorded configuration as the timing workload: $h = 0.01\,\mathrm{s}$, cloth coefficients $(300, 10, 10^4)$, and a 3 s verification. The unchanged later controller is not part of the optimized parameter set.

Figure 3 counts accepted parameter updates. Figure 20 instead follows recorded objective evaluations, including line-search trials where present. Neither plot index is a count of the individual frame adjoint systems. Loss magnitudes are task-specific and are not compared across tasks.

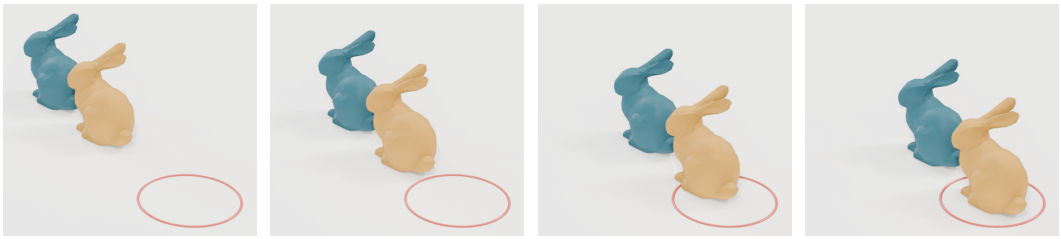
E.2 FULL-SPACE MOTION, DESIGN, AND IDENTIFICATION

The five shared tasks retain full-space FEM coordinates in both systems. They cover losses on another object's final position, on a complete trajectory, on internal stress, and on observed deformation or a prescribed final position. Figure 3 collects their optimization histories; Section E.5 compares the evaluation costs.

Bunny collision. We optimize a shared initial velocity for the vertices of one bunny so that collision moves the other bunny to a target. The objective uses only the second bunny's volume-weighted horizontal center after 40 frames:

$$L_{\text{bunny}} = \|\mathbf{c}_{xz}(\mathbf{x}_T^{(2)}) - \mathbf{c}_{xz}^*\|^2. \tag{101}$$

The loss depends on the first bunny's initial velocity because the collision changes the second bunny's motion. It contains no direct position target for the first bunny. The displayed history shows a lower final loss for DiffGIPC, whereas the DiffIPC trace decreases initially and then fluctuates at a higher level. Figure 10 illustrates the resulting change in motion. The separate resolution study in Section E.6 also considers affine versions of this task.



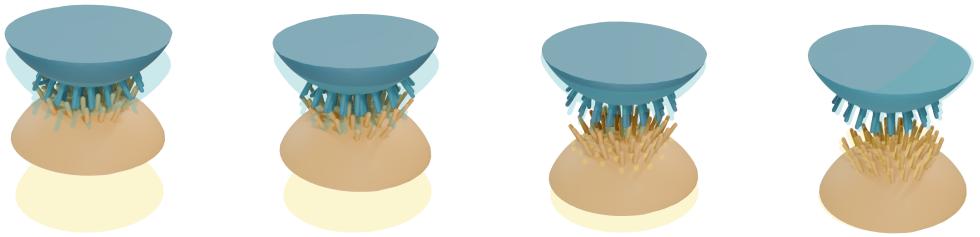
(a) Initial (b) Iteration 1 (c) Iteration 3 (d) Iteration 10

Figure 10: Initial-velocity optimization for two deformable bunnies. The moving bunny is controlled so that the second bunny's final horizontal centroid matches a target. The panels show frame 40 at updates 0, 1, 3, and 10.

Tentacle trajectory matching. The first of two contacting deformable bodies has an optimized shared initial velocity; the second body's initial velocity remains fixed. Unlike the terminal bunny objective, the loss accumulates position error over all 100 frames. With tetrahedral weights ω_i^{tet} and weighted position $\mathbf{c}_t^{\text{tent}} = \sum_i \omega_i^{\text{tet}} \mathbf{x}_{t,i}$, we use

$$L_{\text{tentacle}} = 20h \sum_{t=1}^{T} \|\mathbf{c}_t^{\text{tent}} - \mathbf{c}_t^{\text{tent},*}\|^2. \tag{102}$$

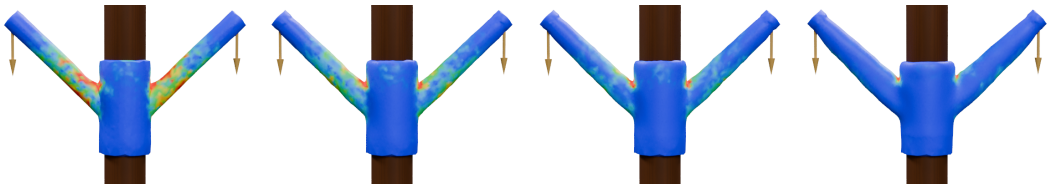
Each output frame supplies a loss gradient, and the reverse pass propagates both motion and friction-history dependencies. The displayed DiffGIPC loss decreases overall and ends below the DiffIPC trace. Figure 11 shows how the optimized motion approaches the prescribed trajectory.



(a) Initial (b) Iteration 1 (c) Iteration 3 (d) Iteration 10

Figure 11: Matching the motion of two contacting deformable bodies by optimizing the first body's initial velocity. Opaque surfaces show simulated bodies; translucent surfaces show the target. The panels show frame 100 at the labeled optimization updates.

Loaded Support design. We optimize selected rest vertices of a loaded support while fixing its supports. Each objective evaluation advances 20 frames. The measured comparison uses the power-2 stress objective, with feasibility checks, backtracking, and remeshing to handle invalid or poor-quality tetrahedra. Both systems reach similar final plotted losses. Remeshing changes the discrete objective, and the displayed histories include upward jumps. The curves therefore do not describe monotone descent of one fixed-mesh objective. The visualization in Figure 12 shows the resulting spatial changes in shape and stress under its stated objective.



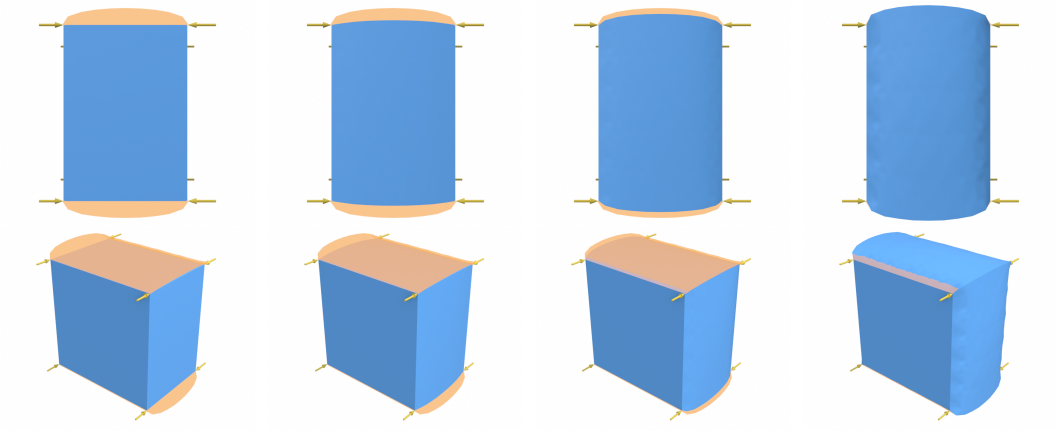
(a) Initial (b) Iteration 20 (c) Iteration 50 (d) Iteration 200

Figure 12: Loaded Support rest-shape design under downward loading with fixed supports. The panels use the power-2 first-Piola stress objective. Panels show the loaded configuration after 20 frames at updates 0, 20, 50, and 200, with one clipped blue-to-red RMS stress scale.

Material identification. For a constrained, loaded cube, we fit two positive material coefficients to observed marker positions. Unconstrained optimizer variables are mapped to the stored coefficients by

$$\theta_{\mathrm{vol}} = e^{z_{\mathrm{vol}}} + \tfrac{5}{6} e^{z_{\mathrm{len}}} + 10^{-4}, \qquad \theta_{\mathrm{len}} = \tfrac{4}{3} e^{z_{\mathrm{len}}}. \tag{103}$$

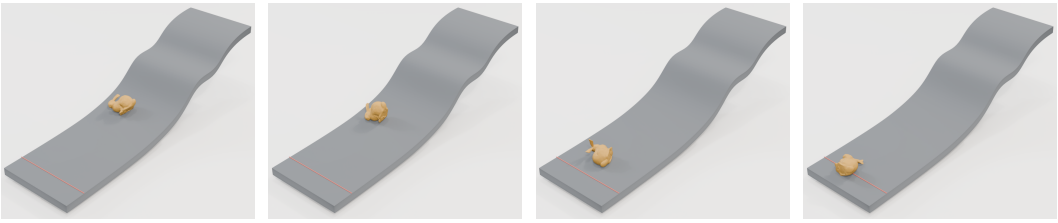
The measured run uses ten static load increments, $h = 0.1$, and input $E = 10^6$. Both systems reduce the marker loss to a similar final level. Figure 13 illustrates the change in loaded shape. The reported loss measures marker-position agreement with the observed deformation, rather than coefficient error.



(a) Iteration 0 (b) Iteration 2 (c) Iteration 5 (d) Iteration 20

Figure 13: Material fitting for a constrained, loaded cube. Two positive material coefficients are optimized to match marker positions. Solid surfaces show the simulation and translucent surfaces show the target; arrows indicate the Dirichlet constraints. The panels show frame 19 at updates 0, 2, 5, and 20, with side and oblique views.

Friction optimization. We optimize a single coefficient shared by object–object and ground friction so that the bunny reaches a prescribed position at a specified time. The objective measures the difference between its simulated position at that time and the target position; it does not compare the coefficient with a prescribed value or match an entire reference trajectory. In the measured experiment, the position is evaluated after 100 frames with $h = 0.02\,\mathrm{s}$. Starting from a coefficient of 0.5, DiffGIPC obtains 0.1, with which the bunny reaches the target position at the specified time. Thus 0.1 is an optimization result, not a prescribed coefficient target. Figure 14 illustrates the task. Both systems reduce the position loss and reach similar final levels at the plotted scale.



(a) Initial (b) Iteration 1 (c) Iteration 3 (d) Iteration 5

Figure 14: Optimizing a shared object–object and ground friction coefficient so that the bunny reaches a target position at the specified time. The panels show the position after 100 frames for successive parameter updates in this visualization. The measured experiment uses 100 frames at $h = 0.02\,\mathrm{s}$; its coefficient changes from 0.5 to the optimized value 0.1.

Optimization behavior. Across these tasks, the reported final losses favor DiffGIPC on bunny collision and tentacle matching, and are similar at the plotted scales on the Loaded Support, material, and friction tasks. The horizontal axes count accepted updates, with zero denoting the initial parameters. Each panel has its own objective and scale. These histories show the outcomes of the two implemented optimizations, not equality of their intermediate states or numerical derivatives.

E.3 Stable placement and rest-shape design

We next consider design and placement tasks beyond the five shared examples. Stone placement changes affine initial configurations without changing shape, cloth fitting changes the deformable rest shape, and Garment Hanger design changes an affine surface in frictional contact with cloth. The corresponding optimization histories appear in Figure 20.

Seven-stone placement. We optimize the horizontal components of seven affine bodies' initial configurations while keeping their shapes fixed. These bodies contribute 84 simulation DoFs. The loss selects the earliest frame with an active centroid-distance penalty. Its tolerance is 0.01 for the first 50 differentiable-simulation updates and 0.15 afterward. The run finishes after 78 parameter updates and records 80 forward and 78 backward evaluations. Figure 15 shows the initial placement and updates 10, 50, and 78, each replayed for 1,000 frames. This task demonstrates placement optimization with detailed surface geometry and compact body coordinates.



(a) Initial placement (b) Iteration 10 (c) Iteration 50 (d) Iteration 78

Figure 15: Stable placement of seven affine bodies with fixed shapes. Only horizontal initial positions are changed. The panels show the initial parameters and differentiable-simulation updates 10, 50, and 78. The first 50 updates use centroid tolerance 0.01 in the loss; later updates use 0.15. Each placement is simulated for 1,000 frames. Every panel shows the last frame of its rendered sequence.

Cloth rest-scale fitting. A single scale parameter changes a garment's rest shape while the mannequin and constraints remain fixed. The measured draping simulation uses 48 steps and stretching, shearing, and bending coefficients $(10^3, 10, 10^4)$. The recorded loss drops rapidly before flattening. The drapes in Figure 16 illustrate the task. This example uses a scalar design map to optimize many rest positions with the same physical derivatives.



(a) Initial (b) Iteration 1 (c) Iteration 3 (d) Iteration 5 (e) Target

Figure 16: Cloth rest-scale fitting on a fixed mannequin. The blue panels show successive drapes and the orange panel shows the target.

Garment Hanger shoulder design. Two Bézier shoulder scales define the rest surface of a garment hanger represented as an affine body and supporting a garment. The objective measures garment support, while collision and friction with coefficient 0.05 connect the cloth motion to the hanger parameters. This task uses the affine rest-shape derivatives, including their dependence on mass, geometry, and friction, without replacing the hanger by a full-space FEM body. The measured optimization selects scales $(0.8285, 0.8000)$. The measured workload contains thirty optimization rollouts that may stop when the garment drops, plus two full 600-frame verification rollouts. The full checks assess retention beyond the optimization rollouts, which may terminate before frame 600. Figure 17a shows a separate initial/optimized design comparison.

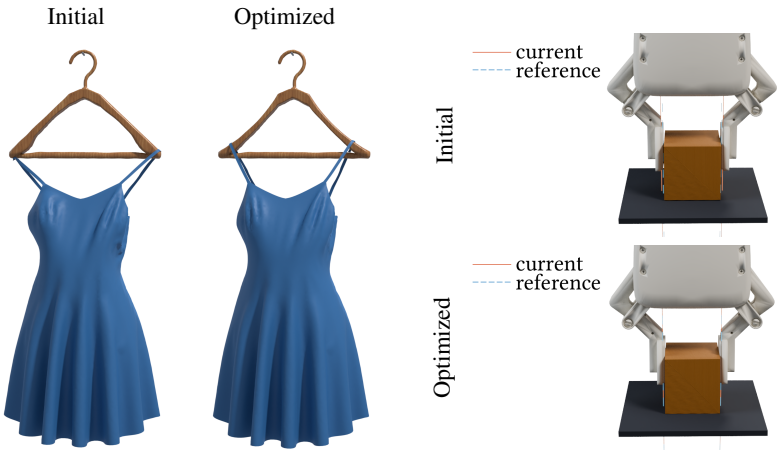
For the Garment Hanger curve, a rollout can stop at garment-drop frame k. To display progress against a common horizon n, the recorded plot adds the final iterate's per-frame losses for frames

$k+1, \ldots, n$ to that rollout's value, then applies a constant offset so that the final plotted value is zero. This is a processed progress measure, not the raw objective of the truncated simulation or evidence of zero physical loss. The actual support check is evaluated by the full 600-frame verification. Section E.6 compares this affine design model with a surface-matched FEM replacement.

E.4 Articulated control

The following tasks use joint targets or a shared gripper aperture in the coupled affine-body system. They distinguish the optimized objective from a later task check: grasp-angle and aperture losses depend on simulated joint coordinates, whereas cup catching uses plate tracking. Grasp or catch success is checked separately from the optimization loss.

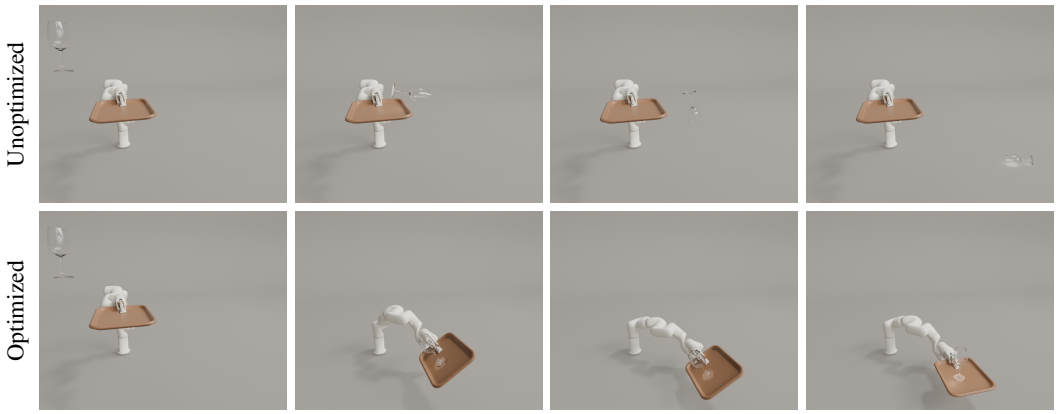
Grasp-angle control. An articulated affine robot approaches and grasps a FEM object above a fixed support. We optimize one shared finger-joint target by comparing four simulated joint angles with 0.45 rad during closure. The 120-frame horizon includes approach, closure, and return motion. The measured loss decreases and then flattens. Figure 17b shows a separate initial/optimized grasp; its dashed image guides are not part of the objective.



(a) Garment Hanger shape design (b) Grasp-angle control

Figure 17: Garment Hanger shape design and joint-target optimization in separate visualizations. (a) Two Bézier shoulder scales are optimized to support a garment under frictional contact. Both views show frame 33, when the garment begins to slip off the initial hanger; the optimized design retains both straps in 600-frame checks. (b) Initial and optimized grasp commands are 0.39 and 0.447843 rad after 20 updates. The objective compares four simulated joint angles with 0.45 rad. Solid coral guides follow the fingers; dashed blue guides are visual references only.

Cup catching. Joint targets move a plate welded to an xArm gripper to catch an incoming affine cup. The loss compares plate position, linear velocity, and affine orientation with a reference, and weakly penalizes differences between consecutive joint targets. Cup motion is used to check catch success but does not appear in the loss. The measured run uses 240 frames at $h = 0.005$ s and friction coefficient 2.0. Its history is nonmonotone, with large intermediate excursions before a lower-loss catching configuration is reached. The run stops at a successful catch after 40 backward evaluations. Figure 18 illustrates the initial and optimized controls. The stopping point is determined by catch success rather than a fixed number of optimization updates.



(a) Frame 94 (b) Frame 111 (c) Frame 120 (d) Frame 239

Figure 18: Articulated cup catching with a plate-tracking objective. The upper and lower rows show zero-target controls and controls after 40 updates. Columns show frames 94, 111, 120, and 239 at $h = 0.005$ s. The cup falls away with the initial command and remains on the plate with the optimized command.

Dual-arm towel grasp. One symmetric aperture supplies the targets of four prismatic finger joints on two Piper arms. The approach motion is fixed, and the loss uses the simulated gripper opening over the first 142 frames. After selecting the command, a 300-frame rollout checks the grasp with the unchanged controller tracking opposite towel corners. The measured run uses $h = 0.01$ s and cloth coefficients $(300, 10, 10^4)$, so this check spans 3 s. The recorded loss decreases. Figure 19 shows frames 0, 170, 210, and 300 from this same 3 s evaluation. Only the aperture is optimized; the later folding controller is fixed.

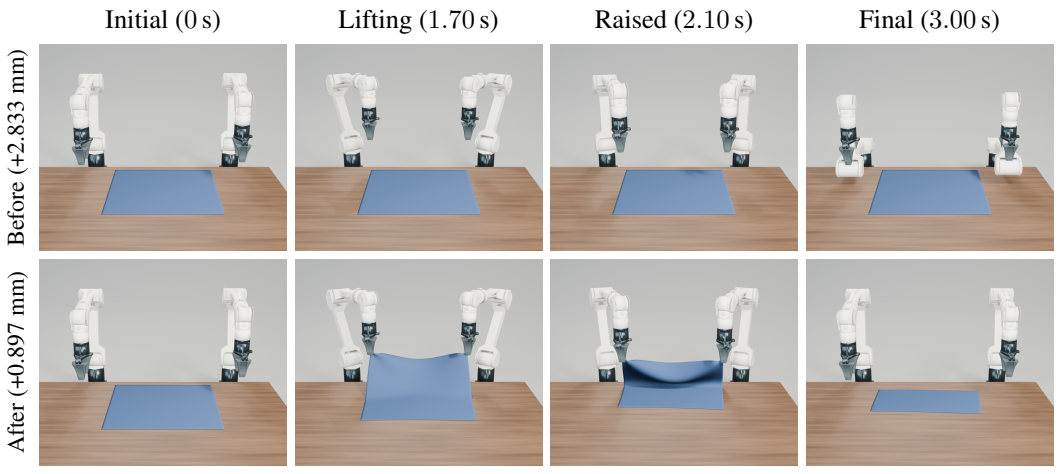
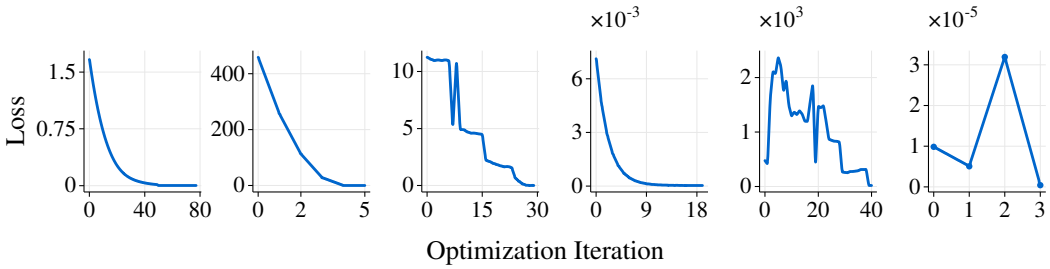


Figure 19: Dual-arm towel grasp with a shared ABD target-height offset. This visualization uses cloth coefficients $(300, 10, 10^4)$ and $h = 0.01$ s. With the optimized command, the robot successfully grasps and folds the cloth. The columns show frames 0, 170, 210, and 300.

The six additional histories in Figure 20 show reductions for placement, cloth scale, grasp angle, and dual-arm target height, with larger intermediate changes in cup catching and Garment Hanger design. These plots count recorded objective evaluations, including line-search trials where present, rather than the accepted updates used for the five shared tasks. Together, the tasks demonstrate reuse of the parameter and loss interface across design, identification, placement, and control, with each task's success criterion stated separately from its optimization loss.



(a) Fig. 15 (b) Fig. 16 (c) Fig. 17a (d) Fig. 17b (e) Fig. 18 (f) Fig. 19

Figure 20: Optimization histories for the six additional tasks. The first five horizontal axes count recorded objective evaluations, including line-search trials where present, rather than native simulation calls. The dual-arm panel shows four Adam corner-height losses. Cup catching ends at the successful catch. The Garment Hanger panel is a processed progress curve: truncated rollouts are completed with the final iterate's remaining per-frame losses, and a constant offset makes the last value zero. This is not a zero physical loss. The processing convention is specified with the Garment Hanger task in Section E.3.

E.5 Comparison with DiffIPC

We compare the mean evaluation costs of the five full-space tasks in Section E.2. Both systems execute complete optimization runs, but the reported values are per-evaluation means, not matched end-to-end optimization times. The systems can visit different states and use different formulations, timer boundaries, and effective stopping criteria. Table 5 reports the forward and backward phase times, their totals, and the corresponding speedups.

Table 5: Mean time per forward and backward evaluation (seconds) on the five shared examples. DiffIPC/PolyFEM uses 32 CPU threads; DiffGIPC uses an RTX 4090. Speedup is computed as the DiffIPC time divided by the DiffGIPC time. The rightmost Total adds the two per-call means, not all calls in an optimization.

Example	System	Forward						Backward				Total
		Linear	CCD	Hessian	Search	Other	Total	Hessian	Linear	Other	Total	
Bunny collision (Fig. 10)	DiffIPC	4.455415	10.310270	4.611112	1.120208	1.918848	22.415853	1.418722	1.090980	0.438421	2.948123	25.363976
	DiffGIPC	0.673068	0.051846	0.039421	0.021800	0.004494	0.790629	0.009008	0.229039	0.036901	0.274948	1.065577
	Speedup	6.62×	198.86×	116.97×	51.38×	427.01×	28.35×	157.49×	4.76×	11.88×	10.72×	23.80×
Tentacle matching (Fig. 11)	DiffIPC	137.210200	14.269903	119.281248	66.085898	11.289078	348.136327	7.351265	11.086580	2.838195	21.276040	369.412367
	DiffGIPC	9.726493	1.223369	1.148023	0.624651	0.031337	12.753873	0.042639	1.041641	0.229404	1.313685	14.067558
	Speedup	14.11×	11.66×	103.90×	105.80×	360.25×	27.30×	172.41×	10.64×	12.37×	16.20×	26.26×
Loaded Support (Fig. 12)	DiffIPC	22.147000	0.597361	3.233300	1.102100	10.873575	37.953336	2.901818	12.476000	0.487006	15.864824	53.818160
	DiffGIPC	0.936785	0.005401	0.014030	0.001850	0.001270	0.959336	0.012022	0.864571	0.039606	0.916199	1.875535
	Speedup	23.64×	110.61×	230.45×	595.75×	8559.83×	39.56×	241.37×	14.43×	12.30×	17.32×	28.69×
Material fitting (Fig. 13)	DiffIPC	0.894114	0.029420	0.685650	0.027724	0.611840	2.248748	0.352779	0.532415	0.057421	0.942615	3.191363
	DiffGIPC	0.015453	0.002244	0.004363	0.000995	0.004871	0.027926	0.003759	0.022587	0.010263	0.036609	0.064534
	Speedup	57.86×	13.11×	157.16×	27.87×	125.61×	80.53×	93.86×	23.57×	5.59×	25.75×	49.45×
Friction optimization (Fig. 14)	DiffIPC	17.418981	22.162291	21.145685	8.142143	6.849071	75.718170	2.462017	1.924994	1.239815	5.626826	81.344996
	DiffGIPC	3.719368	1.310757	1.285420	0.688029	0.042479	7.046053	0.046313	0.364963	0.207919	0.619195	7.665248
	Speedup	4.68×	16.91×	16.45×	11.83×	161.24×	10.75×	53.16×	5.27×	5.96×	9.09×	10.61×

The recorded forward-plus-backward speedups over DiffIPC range from $10.61\times$ for friction optimization to $49.45\times$ for material fitting. Bunny collision, tentacle matching, and Loaded Support design give $23.80\times$, $26.26\times$, and $28.69\times$, respectively. Backward-evaluation speedups range from 9.09 to $25.75\times$, while forward-evaluation speedups range from 10.75 to $80.53\times$. The backward totals measure the complete reported differentiation call, not only its linear solver; the combined totals also include the StiffGIPC-based forward computation.

The phase records show lower mean times for every listed DiffGIPC phase. They help locate the costs, but do not isolate individual algorithmic improvements. In particular, the *Other* category is computed by subtracting the named phases from a separately measured total, and need not contain equivalent work in the two systems. We use the directional totals and the optimization histo-

ries together when interpreting the comparison, rather than interpreting the largest individual phase speedup as an isolated kernel speedup.

E.6 Effect of object representation and resolution

The preceding comparison keeps full-space FEM in both systems. We now examine the additional effect of affine coordinates using a bunny-resolution study and a Garment Hanger comparison. Table 6 reports the forward and backward phase times for each model. The rightmost total adds one mean forward and one mean backward evaluation.

Table 6: Mean forward and backward evaluation times (seconds) for the bunny-collision resolution study and Garment Hanger rest-shape design. The rightmost Total is the sum of the mean forward and mean backward times, not accumulated optimization time. Each bunny representation generates its own target; physical responses can differ across representations.

Resolution / case	System	Forward						Backward				Total
		Linear	CCD	Hessian	Search	Other	Total	Hessian	Linear	Other	Total	
Bunny collision (Fig. 10): resolution study												
933 vertices	DiffGIPC ABD+ABD	0.025698	0.037918	0.039585	0.021079	0.003822	0.128102	0.008353	0.004271	0.040604	0.053228	0.181329
	DiffGIPC ABD+FEM	0.552937	0.042148	0.053053	0.029102	0.003858	0.681097	0.011575	0.219918	0.053582	0.285075	0.966172
	DiffGIPC FEM+FEM	0.673068	0.051846	0.039421	0.021800	0.004494	0.790629	0.009008	0.229039	0.036901	0.274948	1.065577
	DiffIPC FEM+FEM	4.455415	10.310270	4.611112	1.120208	1.918848	22.415853	1.418722	1.090980	0.438421	2.948123	25.363976
7,991 vertices	DiffGIPC ABD+ABD	0.031878	0.052560	0.102927	0.027149	0.004726	0.219239	0.019382	0.004801	0.070357	0.094541	0.313780
	DiffGIPC ABD+FEM	1.399342	0.073773	0.183370	0.048374	0.006117	1.710976	0.035651	0.321428	0.121896	0.478975	2.189952
	DiffGIPC FEM+FEM	2.215480	0.083541	0.187218	0.042943	0.006094	2.535277	0.031703	0.502727	0.082871	0.617302	3.152578
	DiffIPC FEM+FEM	79.616448	23.650845	53.596786	9.720479	7.600858	174.185416	15.149235	16.955800	0.983365	33.088400	207.273816
28,435 vertices	DiffGIPC ABD+ABD	0.028574	0.058501	0.206672	0.026168	0.005145	0.325060	0.043349	0.004429	0.145764	0.193542	0.518602
	DiffGIPC ABD+FEM	3.454463	0.084599	0.515464	0.055802	0.005629	4.115957	0.101424	0.951931	0.303122	1.356477	5.472434
	DiffGIPC FEM+FEM	6.859357	0.078042	0.708793	0.049181	0.004157	7.699529	0.112351	2.153385	0.226633	2.492370	10.191899
	DiffIPC FEM+FEM	741.018286	58.448005	437.204357	54.672234	32.175647	1323.518529	99.600996	112.670000	4.620404	216.891400	1540.409929
Garment Hanger (Fig. 17a): rest-shape optimization												
Bezier pair	DiffGIPC ABD	3.998944	1.088863	1.665408	1.338391	0.028664	8.120270	0.050185	0.218725	0.121809	0.390719	8.510989
	DiffGIPC FEM	10.722439	0.922330	1.504406	0.630590	0.019467	13.799233	0.035638	0.913963	0.095372	1.044974	14.844206

Table 7: Complete optimization workloads and mean phase times (seconds) of the eleven examples. N_f and N_b count complete forward and backward evaluations, respectively, including the applicable trial and verification calls, not individual frame systems. Phase columns and the forward/backward Totals are per-call means. The rightmost Total sums the times of all forward and backward calls, excluding setup and export.

#	Example	N_f	N_b	Forward mean						Backward mean				Total
				Linear	CCD	Hessian	Search	Other	Total	Hessian	Linear	Other	Total	
1	Stone placement (Fig. 15)	80	78	0.114	0.086	0.133	0.026	0.011	0.370	0.054	0.075	0.273	0.402	60.91
2	Bunny collision (Fig. 10)	14	11	0.673	0.052	0.039	0.022	0.004	0.791	0.009	0.229	0.037	0.275	14.09
3	Tentacle matching (Fig. 11)	11	10	9.726	1.223	1.148	0.625	0.031	12.754	0.043	1.042	0.229	1.314	153.43
4	Garment Hanger (Fig. 17a)	32	30	3.999	1.089	1.665	1.338	0.029	8.120	0.050	0.219	0.122	0.391	271.57
5	Loaded Support (Fig. 12)	233	200	0.937	0.005	0.014	0.002	0.001	0.959	0.012	0.865	0.040	0.916	406.77
6	Material fitting (Fig. 13)	21	20	0.015	0.002	0.004	0.001	0.005	0.028	0.004	0.023	0.010	0.037	1.32
7	Friction optimization (Fig. 14)	5	5	3.719	1.311	1.285	0.688	0.042	7.046	0.046	0.365	0.208	0.619	38.33
8	Cloth rest scale (Fig. 16)	7	6	17.799	0.826	1.073	0.923	0.032	20.654	0.004	0.618	0.005	0.627	148.34
9	Grasp angle (Fig. 17b)	44	20	0.395	0.080	0.182	0.073	0.013	0.744	0.048	0.076	0.237	0.360	39.94
10	Cup catching (Fig. 18)	41	40	2.466	1.141	1.195	0.564	0.052	5.418	0.181	0.660	1.298	2.139	307.72
11	Dual-arm grasp (Fig. 19)	11	5	4.770	2.752	4.092	1.131	0.044	12.789	0.202	0.770	1.761	2.733	154.35

Bunny resolution study. FEM+FEM uses two deformable bodies, ABD+FEM uses one affine body and one FEM body, and ABD+ABD uses two affine bodies. Each representation generates its own target. The comparison therefore includes different physical models and reference trajectories, not only different numbers of motion DoFs. As resolution grows from 933 to 28,435 vertices, mean forward-plus-backward time increases from 1.066 to 10.192 s for DiffGIPC FEM+FEM and from

0.181 to 0.519 s for ABD+ABD. The increases are about $9.56\times$ and $2.86\times$, respectively. The affine cost still grows because the surface geometry is resolved; it is not independent of mesh resolution.

At the highest resolution, ABD+ABD is about $19.65\times$ faster than DiffGIPC FEM+FEM in this measurement. The backward linear phase falls from 2.153385 to 0.004429 s, while the total backward time falls from 2.492370 to 0.193542 s. These phase times show that assembly and gradient accumulation remain significant after the linear-system cost is reduced. The forward-plus-backward evaluation speedups of DiffGIPC FEM+FEM over DiffIPC FEM+FEM increase from $23.80\times$ to $151.14\times$ across the resolutions. Comparing DiffIPC FEM+FEM with DiffGIPC ABD+ABD at the highest resolution yields $2970.31\times$. This last speedup combines a change in implementation with a change in representation; it is not GPU acceleration alone.

Garment Hanger: ABD versus FEM. The two models retain 4,476 surface vertices and 8,952 triangles. The FEM replacement adds 902 interior vertices and 18,256 tetrahedra; a harmonic extension maps the two Bézier shape fields to the interior. The mean forward-plus-backward times are 8.511 s for ABD and 14.844 s for FEM, giving the ABD model a speedup of about $1.74\times$. The backward totals are 0.390719 and 1.044974 s. The material definitions and rest-update rules differ, FEM permits non-affine deformation, and the volume mesh contains low-quality tetrahedra. These results measure the cost of the implemented model choices, not coordinate count with all other factors fixed. Separately measured full-process optimization times are 280.006 s for ABD and 478.595 s for FEM. These process-level times are not the accumulated evaluation time in Table 7.

E.7 Forward and backward workload

Table 7 places the comparisons in the context of all eleven optimizations. Accumulated evaluation times range from 1.32 s for material fitting to 406.77 s for Loaded Support design. Different tasks require different numbers of calls: for example, Loaded Support design uses 233 forward evaluations and 200 backward evaluations, whereas grasp-angle optimization uses 44 and 20. The cost of one forward-plus-backward pair alone does not describe these complete recorded workloads.

The phase breakdown in Table 7 also varies substantially across tasks. Forward linear systems account for 17.799 of 20.654 s per cloth-scale call, about 86%. For dual-arm grasping, forward linear solves, collision detection, and Hessian assembly take 4.845, 3.122, and 4.371 s, respectively. In the backward pass, the Loaded Support linear phase takes 0.865 of 0.916 s, whereas the dual-arm *Other* category takes 1.996 of 2.840 s. These measurements show why a reduction in linear-system time need not produce the same speedup for a complete evaluation.

Garment Hanger forward averages mix thirty cutoff-limited rollouts with two 600-frame checks; dual-arm averages mix thirteen 142-frame calls with two 300-frame checks. The reported means must be interpreted with these horizons and call counts.