Multi-level Quadrature for Projective Dynamics

DEWEN GUO*, University of Utah, USA
YUQI MENG*, University of Utah, USA
ZIXUAN LU*, University of Utah, USA
ZHIYONG HE, University of Utah, USA
HUAMIN WANG, Style3D Research, China
LEI LAN, Zhejiang University, China
WEIWEI XU, Zhejiang University, China
KUI WU, LIGHTSPEED, USA
CHENFANFU JIANG, UCLA, USA
YIN YANG, University of Utah, USA

Fig. 1. This paper introduces a new integration-based parallel simulation framework leveraging projective dynamics (PD). It is well-known that PD converts nonlinear elastodynamics simulation into a local-global optimization. While the local solve is highly parallelizable, the global solve in PD has always been the bottleneck. Even with pre-factorization, the forward and backward substitution of the global matrix remains costly for high-resolution models. We approach the fast and parallel PD global solve from the perspective of integration: the exact global solve at each nodal unknown is the quotient between two volumetric integrals. Based on this observation, we design a multi-level numerical quadrature algorithm aiming to accurately estimate the near-, middle- and far-field integration sparsely. The teaser figure showcases a representative example. The branches of the tree consist of nearly 800K DOFs, with 23.5K leaves appended to the branches. We use an auxiliary voxel grid to generate integration domains (as highlighted). Our new PD solver simulates this model interactively at 95 ms per time step ($\Delta t = 1/100$ s).

*equal contribution

Authors' Contact Information: Dewen Guo, University of Utah, Salt Lake City, USA, guodw.sh@gmail.com; Yuqi Meng, University of Utah, Salt Lake City, USA, aressegetesstery@gmail.com; Zixuan Lu, University of Utah, Salt Lake City, USA, birdpeople1984@gmail.com; Zhiyong He, University of Utah, Salt Lake City, USA, zhiyong.he@utah.edu; Huamin Wang, Style3D Research, Hangzhou, China, wanghmin@gmail.com; Lei Lan, Zhejiang University, Hangzhou, China, lanlei.virhum@gmail.com; Weiwei Xu, Zhejiang University, Hangzhou, China, xww@cad.zju.edu.cn; Kui Wu, LIGHTSPEED, Los Angeles, USA, kwwu@lightspeed-studios.com; Chenfanfu Jiang, UCLA, Los Angeles, USA, chenfanfu.jiang@gmail.com; Yin Yang, University of Utah, Salt Lake City, USA, yangzzzy@gmail.com.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2026 Copyright held by the owner/author(s).
ACM 1557-7368/2026/12-ART218
https://doi.org/10.1145/3842501

Projective dynamics (PD) is a popular framework for efficient physical simulation. However, its factorization-based global solve maps poorly to GPU parallelization, while iterative alternatives often fail to converge effectively for stiff or high-resolution models. This paper reformulates the PD global solve by observing that its primary computational bottleneck lies in vector dot products, which can be reinterpreted as discrete integrations over the deformable body. The resulting integrand exhibits a Yukawa-like kernel structure that peaks sharply at each mesh node and decays exponentially with distance. Exploiting this prior, we partition the domain into near, middle, and far fields around each node, and design a quadrature rule for each that matches the local behavior of the integrand: an anisotropic Gaussian fit with a closed-form integral in the near field, a low-order Gauss-Legendre rule for the smoothed residual in the middle field, and a weighted k-means++ clustering with centroid-based quadrature in the irregular far field. This integration perspective reveals an effective sparsity pattern that enables both excellent parallelism and rapid convergence. Our algorithm also serves as a strong preconditioner for collision-involved PD systems and delivers

significant speedup over existing PD solvers across a range of deformable scenarios.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: Numerical integration, Optimization, Projective dynamics, Deformable model

ACM Reference Format:
Dewen Guo, Yuqi Meng, Zixuan Lu, Zhiyong He, Huamin Wang, Lei Lan, Weiwei Xu, Kui Wu, Chenfanfu Jiang, and Yin Yang. 2026. Multi-level Quadrature for Projective Dynamics. *ACM Trans. Graph.* 45, 6, Article 218 (December 2026), 17 pages. https://doi.org/10.1145/3842501

1 Introduction

Projective dynamics (PD) [Bouaziz et al. 2014] has emerged as a popular framework for deformable simulation. It reformulates the optimization into global-local alternating iterations, where the local step is fully parallelizable. In the global step, the system matrix remains constant as long as the constraint set is fixed, and can be pre-factored. This design effectively avoids the overhead of repeated matrix factorizations compared with Newton-based algorithms, making PD a favored choice for interactive physics-based animation.

While the global matrix in PD may be pre-factored, the forward and backward substitutions remain sequential and do not scale well on the GPU as the total number of degrees of freedom (DOFs) grows. As a result, the simulation quickly becomes prohibitive if multiple PD iterations are needed. To address this, several GPU-based schemes have been proposed to solve the global system *inexactly*, replacing the matrix factorization with parallelizable iterative processes. Representative approaches include Chebyshev acceleration [Wang 2015], colored Gauss-Seidel (GS) [Fratarcangeli et al. 2016], aggregated Jacobi [Lan et al. 2022], and quasi-Newton PD methods [Li et al. 2023]. These methods suffer from poor convergence when dealing with high-resolution models or stiffer scenarios.

In this paper, we revisit the procedure of the PD global solve. While it appears to be a straightforward matrix inversion, we observe that its core computation, and the primary bottleneck, lies in vector dot products. Interpreting the nodal values of a vector as samples from an underlying continuous field, such a dot product can be viewed as an integral over the entire simulation domain. Building on this observation, we design a new parallel algorithm similar to the classic Jacobi process, in which each per-node subproblem reduces to evaluating one such integral. By approximating these integrals with a tailored sparse quadrature, a single Jacobi-like iteration of our solver yields results comparable to those from a full matrix factorization.

The key to our accelerated convergence is a sparse approximation of the per-node integral in each Jacobi-like subproblem, where the integral can be viewed as the response of the deformable body to a unit impulse applied at the node. As the PD global matrix closely resembles a shifted Laplacian operator, its impulse response admits a Yukawa-like Green's function, which peaks sharply at the node and decays exponentially with distance. Exploiting this structure, we partition the simulation domain into the near, middle, and far fields relative to the node, and design a quadrature rule for each that matches the local behavior of the integrand. In the near field, where the integrand inherits a sharp exponential peak, we fit a small anisotropic Gaussian surrogate and use its integral as an approximation. In the middle field, taken as a larger axis-aligned box enclosing the near field, the residual after removing the exponential peak exhibits only smooth variation, so a low-order Gauss-Legendre rule suffices. In the far field, the geometrically irregular remainder outside the box, we precompute a weighted k-means++ clustering of its connected components and emit a single centroid-based quadrature point per cluster.

Algebraically, our method acts as a sparsification strategy for dot products. However, it is only by interpreting the problem through the perspective of integration that an effective sparsity pattern can be forged. In scenarios where collision-free systems can be resolved in just one or two Jacobi-like iterations, our approach also serves as a strong preconditioner for the system matrix during collisions, giving excellent convergence under general setups. We implemented our framework and evaluated it across various deformable simulation scenarios, where our method consistently achieves significant speedup over existing PD-based algorithms.

2 Related work

Simulating deformable bodies via discretizations of continuum mechanics has attracted significant interest in the graphics community since the pioneering work by Terzopoulos et al. [1987]. Subsequent research has explored a wide range of topics, including nonlinear elasticity models based on Finite Element Method (FEM) [Kikuuwe et al. 2009; Sifakis and Barbič 2012], mass-spring systems for efficient and interactive cloth simulation [Baraff and Witkin 1998; Provot 1995], meshless and particle-based methods for handling large deformations [Desbrun and Gascuel 1996; Guo et al. 2025a], propagation of fracture [O'Brien and Hodgins 1999; Shen and Yang 1998], as well as reduced-order and modal analysis techniques [Barbič and James 2005; Choi and Ko 2005; Pentland and Williams 1989]. These approaches have been widely applied to character animation, digital fashion design [Guo et al. 2025b,c], interactive simulation, and related domains [Joshi et al. 2007; Waters 1987]. Time integration strategies evolve from explicit Euler schemes, which have no stability guarantee under large timesteps [Press et al. 2007], to the unconditionally stable implicit Euler methods. Instead of directly deriving forces, Martin et al. [2011] proposed an equivalent non-convex optimization formulation using elastic potentials. This optimization-based time integration framework has been extended to handle contact by incorporating a barrier energy term [Li et al. 2020], and its computational performance has been significantly improved through GPU acceleration [Guo et al. 2024, 2026b].

A class of constraint-based methods emerged after the foundational position-based dynamics (PBD) [Müller et al. 2007]. PBD reformulates physical simulation by replacing internal forces with geometric constraints that are directly enforced on particle positions. Simulation is advanced through an explicit Euler prediction step followed by iterative constraint projections, avoiding force integrations and global linear system solves, which are often bottlenecks of implicit-Euler-based methods. This design yields high computational efficiency and robust stability under large time steps, making PBD well suited for real-time applications. However, classical PBD is largely heuristic and exhibits iteration-dependent stiffness, where

material rigidity is implicitly controlled by solver iterations and time step size. Extended PBD (XPBD) [Macklin et al. 2016] addresses this limitation by introducing compliance parameters that endow constraints with physical meaning and eliminate iteration dependence. Building on XPBD, subsequent extensions such as XPBI [Yu et al. 2024] and MGPBD [Li et al. 2025] further expand the expressiveness of constraint-based solvers, by incorporating plasticity and accelerating convergence via a multi-grid solver.

Building on optimization-based timestepping with a constraint-based energy formulation, *projective dynamics* (PD) [Bouaziz et al. 2014] introduces an efficient simulation framework that tackles the nonlinear optimization problem by alternating local constraint projections and global quadratic minimization. This optimization structure endows PD with two key computational advantages: first, the local solves are independent across constraints and thus trivially parallelizable; second, the global quadratic surrogate energy induces a constant Hessian, which can be exploited for efficient linear solves via prefactorization, or effective preconditioning in preconditioned conjugate gradient (PCG) methods. Leveraging these properties, several parallelized PD implementations have been developed, for example using Jacobi-based Chebyshev iterations [Wang 2015], or using Gauss-Seidel iterations with a DOF partition [Fratarcangeli et al. 2016]. Subsequent works have further extended PD to support additional energy models [Overby et al. 2017], incorporate model reduction for real-time simulation [Brandt et al. 2018], or improve parallelism on CPU [He et al. 2026; Lu et al. 2025] or GPU [Yuan et al. 2026] through domain decomposition techniques.

Various extensions of PD have focused on robust collision and contact handling, which is challenging as the addition or removal of contact pairs changes the active constraints in the global quadratic system. To address this problem, Ly et al. [2020] introduces an integration scheme for non-penetration and dry friction constraints assuming contacts occur only at nodes. To ensure penetration-free simulation while maintaining efficiency, Lan et al. [2022] integrates incremental potential contact (IPC), a recent barrier-based contact handling formulation, and employs an aggregated Jacobi (A-Jacobi) solver for faster global solves. Li et al. [2023] further optimizes this formulation by designing an effective subspace preconditioner for cloth simulation.

In parallel to PD, another line of work investigates *model reduction* as an alternative approach to efficient simulation. Rather than simplifying energy formulation via a quadratic surrogate, model reduction projects the global system matrix onto low-dimensional subspaces that preserve the dominant deformation modes [Barbič and James 2005; Choi and Ko 2005; Pentland and Williams 1989]. Although solving within the subspace significantly reduces system size, the projection onto the subspace still requires integration over all elements. Moreover, an inappropriate choice of subspace could also introduce severe artifacts such as locking or spurious high-frequency responses, as the deformation is constrained to a low-dimensional subspace, where geometric nonlinearity cannot be fully captured [Guo and Rega 2023]. Constructing suitable subspaces, for example using linear modal analysis, can also incur substantial precomputation cost.

Several works have addressed these limitations. To avoid costly integration when projecting onto the subspace, and ensure that the integration scheme works for arbitrary mesh domains where Gaussian quadrature does not necessarily apply [Hughes 2000], An et al. [2008] introduces cubature sampling, which uses a data-driven greedy algorithm to select a small subset of elements (cubature points) and determine their associated weights for approximation of reduced energy Hessian. Geometric nonlinearity can be better captured using modal derivatives [Barbič and James 2005], geometric modal derivatives [von Tycowicz et al. 2013], or dynamic subspace adaptation [Teng et al. 2015]. Precomputation cost can also be reduced by a careful selection of training deformations [Yang et al. 2015]. Some prior works seek purely local update schemes that combine the parallel local-solve structure of PD with the reduced per-iteration cost characteristic of model reduction. Vertex Block Descent (VBD) [Chen et al. 2024] performs per-vertex Gauss-Seidel (GS) updates to directly minimize the global variational energy, while its augmented Lagrangian extension (AVBD) [Giles et al. 2025] improves robustness in the presence of constraints. More recently, JGS2 [Guo et al. 2026a; Lan et al. 2025] introduces a corotated subspace formulation, in which per-node rigid rotations are factored out, and local updates are applied in rotation-aligned coordinates. Combined with a cubature-based sparse integration scheme, this approach enables near-second-order convergence while retaining high parallel efficiency.

3 Preliminaries

We kick off our discussion with a brief review of PD and Gauss-Legendre quadrature, as they serve as the backbone of our framework.

3.1 Projective dynamics

Given a time integration scheme such as implicit Euler, the elastodynamics simulation can be formulated as a variational optimization at each time step such that:

$$\arg\min_{\boldsymbol{q}} E = \frac{1}{2\Delta t^2}\|\boldsymbol{M}^{\frac{1}{2}}(\boldsymbol{q}-\boldsymbol{q}^*)\|_F^2 + \Psi(\boldsymbol{q}), \tag{1}$$

where $\boldsymbol{q} \in \mathbb{R}^{N\times 3}$ collects the unknown position vector along x, y, and z directions for the next time step. N denotes the total number of nodes on the model. $\boldsymbol{q}^* = \hat{\boldsymbol{q}} + \Delta t\hat{\dot{\boldsymbol{q}}} + \Delta t^2\boldsymbol{M}^{-1}\boldsymbol{f}_{ext}$ is a known variable based on the previous nodal position $\hat{\boldsymbol{q}}$ and velocity $\hat{\dot{\boldsymbol{q}}}$. $\boldsymbol{f}_{ext} \in \mathbb{R}^{N\times 3}$ is the known external force. $\boldsymbol{M}$ is the lumped mass matrix, and Δt the time step size. $\Psi(\boldsymbol{q})$ is the elastic potential.

PD approaches the nonlinear optimization of Eq. (1) in a local-global alternating manner, i.e., LG iterations, which assumes Ψ can be approximated as a collection of quadratic constraint measures such that $\Psi(\boldsymbol{q}) = \frac{1}{2}\sum_j w_j||\boldsymbol{A}_j\boldsymbol{S}_j\boldsymbol{q} - \boldsymbol{B}_j\boldsymbol{p}_j||^2$. Here, $\boldsymbol{S}_j$ is a selection matrix picking nodes pertaining to the j-th constraint C_j from $\boldsymbol{q}$. $\boldsymbol{A}_j$ and $\boldsymbol{B}_j$ are linear operators that map the positional DOFs of $\boldsymbol{q}_j = \boldsymbol{S}_j\boldsymbol{q}$ and $\boldsymbol{p}_j$ to the specific coordinates that C_j measures. $\boldsymbol{p}_j$ is the slack variable of C_j, i.e., the projection of $\boldsymbol{q}$ onto C_j's manifold. w_j is the strength of the constraint. This assumption leads to the local step problem of:

$$\arg\min_{\boldsymbol{p}_j} \frac{w_j}{2}\|\boldsymbol{A}_j\boldsymbol{S}_j\boldsymbol{q} - \boldsymbol{B}_j\boldsymbol{p}_j\|_F^2, \text{ s.t. } C_j(\boldsymbol{p}_j) = 0. \tag{2}$$

It is noted that Eq. (2) can be trivially parallelized on the GPU as each constraint C_j is processed independently. Therefore, the local step serves as the primary driver of performance for PD.

After all the slack variables $\boldsymbol{p}_j$ are determined, the global step solves for $\boldsymbol{q}$ through $\nabla_{\boldsymbol{q}} E = 0$, which forms a linear system of:

$$\underbrace{\left(\frac{\boldsymbol{M}}{\Delta t^2} + \sum_j w_j \boldsymbol{S}_j^\top \boldsymbol{A}_j^\top \boldsymbol{A}_j \boldsymbol{S}_j\right)}_{\boldsymbol{H}} \boldsymbol{q} = \underbrace{\frac{\boldsymbol{M}}{\Delta t^2}\boldsymbol{q}^* + \sum_j w_j \boldsymbol{S}_j^\top \boldsymbol{A}_j^\top \boldsymbol{B}_j \boldsymbol{p}_j}_{\boldsymbol{b}}. \quad (3)$$

Assuming the constraint set of the system is unchanged, $\boldsymbol{H} \in \mathbb{R}^{N\times N}$ is a constant and SPD (symmetric positive definite) matrix, and can be pre-factorized. It is known that $\boldsymbol{I}_3 \otimes \boldsymbol{H}$ functions as a good approximation of $\nabla^2 E$ ($\boldsymbol{I}_3$ is the 3×3 identity matrix), and therefore PD can also be considered as a variation of the quasi-Newton family [Liu et al. 2017]. For large-scale simulations, solving Eq. (3) with a pre-factorized $\boldsymbol{H}$ remains expensive because forward and backward substitutions are sequential.

3.2 3D Gauss-Legendre quadrature

Gauss-Legendre (GL) quadrature is a classical numerical integration rule that constitutes an important component for the sparse integration scheme developed later. It estimates definite integrals over an interval, i.e., $[-1, 1]$ using a set of weighted samples: $\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum w_k f(x_k)$. If $f(x)$ can be well approximated by a polynomial of degree n, i.e., $f(x) \approx P_n(x)$, Gauss-Legendre quadrature stands out as one of the best performing numerical solutions for integrating $f(x)$.

The quadrature nodes x_k are the roots of *Legendre polynomials*: $L_n(x) = \frac{1}{2^n n!}\frac{\mathrm{d}^n}{\mathrm{d}x^n}\left(x^2-1\right)^n$, and $w_k = \frac{2}{\left(1-x_k^2\right)\left[L_n'(x_k)\right]^2}$ are the corresponding weights. Since the Legendre polynomial of degree n is orthogonal to any polynomial of degree smaller than n, an n-point rule *exactly* integrates polynomials of degree up to $2n-1$. Promoting 1D integration to 3D is straightforward with Gauss-Legendre quadrature. Consider the integration of a polynomial $P(x, y, z)$ over a 3D rectangular domain of $D = [a_x, b_x] \times [a_y, b_y] \times [a_z, b_z]$. By applying an affine change of variables in each direction, the integral can be mapped to the reference cube $[-1, 1]^3$, and approximated via the tensor product of the 1D rule as:

$$\begin{aligned}\int_D P(x,y,z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z &= J\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}\bar{P}(x,y,z)\,\mathrm{d}x\mathrm{d}y\mathrm{d}z \\ &\approx J\sum_{k_x=1}^{n_x}\sum_{k_y=1}^{n_y}\sum_{k_z=1}^{n_z} w_{k_x} w_{k_y} w_{k_z}\,\bar{P}(x_{k_x}, y_{k_y}, z_{k_z}), \quad (4)\end{aligned}$$

where $\bar{P}$ denotes the integrand expressed in the reference coordinates, and we have $J = \frac{1}{8}(b_x - a_x)(b_y - a_y)(b_z - a_z)$.

4 Second-order Jacobi for PD

We now focus on the global system of $\boldsymbol{Hq} = \boldsymbol{b}$. Here, we consider $\boldsymbol{q}$ and $\boldsymbol{b}$ are $N \times 1$ vectors representing a respective column in Eq. (3) and partition this system block-wise as:

$$\begin{bmatrix} H_i & \boldsymbol{h}_i^\top \\ \boldsymbol{h}_i & \boldsymbol{H}_i \end{bmatrix}\begin{bmatrix} q_i \\ \boldsymbol{q}_i \end{bmatrix} = \begin{bmatrix} b_i \\ \boldsymbol{b}_i \end{bmatrix}, \quad (5)$$

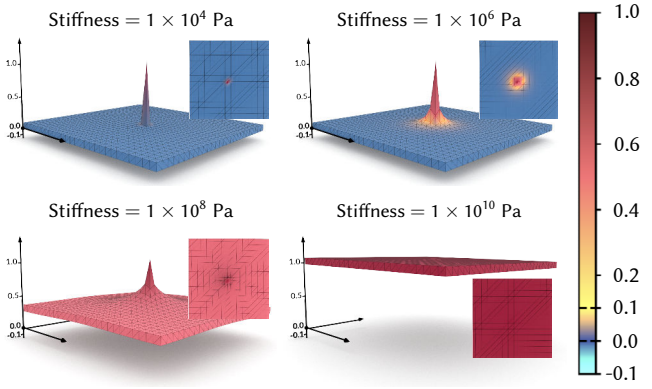

Fig. 2. In our formulation, $\boldsymbol{u}(i)$ encodes the global system response under a unit displacement enforced at node i, i.e., $\boldsymbol{u}(i) = [1, -\boldsymbol{h}_i^\top \boldsymbol{H}_i^{-1}]^\top$. This response exhibits a kernel-like profile, providing a principled basis for strategically sampling the integral $\int u_i(\boldsymbol{x})b(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$.

where H_i, q_i, and b_i are all scalars associated with the i-th node. Meanwhile, $\boldsymbol{H}_i$, $\boldsymbol{q}_i$, and $\boldsymbol{b}_i$ are the corresponding matrix and vector blocks for all the other nodes. $\boldsymbol{h}_i \in \mathbb{R}^{N-1}$ is a column vector.

Expanding the second line of Eq. (5) yields: $\boldsymbol{q}_i = \boldsymbol{H}_i^{-1}(\boldsymbol{b}_i - \boldsymbol{h}_i q_i)$. Substituting it back into the first line of the equation gives us:

$$\left(H_i - \boldsymbol{h}_i^\top \boldsymbol{H}_i^{-1}\boldsymbol{h}_i\right)q_i = -\boldsymbol{h}_i^\top \boldsymbol{H}_i^{-1}\boldsymbol{b}_i + b_i. \quad (6)$$

The coefficient $\left(H_i - \boldsymbol{h}_i^\top \boldsymbol{H}_i^{-1}\boldsymbol{h}_i\right)$ is exactly the Schur complement of $\boldsymbol{H}_i$ in $\boldsymbol{H}$, and Eq. (6) solves for q_i by eliminating all the remaining nodes from the global system. The name *Jacobi* therefore refers to the parallel, per-node structure of the update rather than to an iterative relaxation: unlike a Jacobi iteration, Eq. (6) is exact. Defining $\boldsymbol{u}(i) \in \mathbb{R}^N \triangleq [1, -\boldsymbol{h}_i^\top \boldsymbol{H}_i^{-1}]^\top$ and reorganizing the terms, Eq. (6) can be written as: $\left(H_i - \left(\boldsymbol{h}_i^\top \boldsymbol{H}_i^{-1}\right)\boldsymbol{H}_i\left(\boldsymbol{H}_i^{-1}\boldsymbol{h}_i\right)\right)q_i = -\left(\boldsymbol{h}_i^\top \boldsymbol{H}_i^{-1}\right)\boldsymbol{b}_i + b_i$, or simply:

$$q_i = \frac{\boldsymbol{u}(i)^\top \boldsymbol{b}}{\boldsymbol{u}(i)^\top \boldsymbol{H}\boldsymbol{u}(i)}. \quad (7)$$

The above equation forms the baseline for our parallel solver, as one can verify that the resulting q_i is identical to the solution of the direct solve. Since $\boldsymbol{u}(i)$ can be pre-computed for each node i, $\boldsymbol{u}(i)^\top \boldsymbol{H}\boldsymbol{u}(i)$ can also be pre-computed (if the constraint set remains unchanged). However, computing the inner product of $\boldsymbol{u}(i)^\top \boldsymbol{b}$ at runtime imposes practical challenges. $\boldsymbol{u}(i)$ is a dense vector, and retrieving $\boldsymbol{u}(i)^\top \boldsymbol{b}$ requires traversing all the nodes. For the global solve, we need to perform such a traversal for every q_i. As a result, the computational overhead of Eq. (7) is excessively heavy, making it less helpful for high-resolution PD simulations.

5 From inner product to integration

In this paper, we revisit the inner product computation from an integration perspective by regarding a vector on a 3D model as a discrete representation of a continuous function sampled at nodal points. Without prior knowledge of the function, a standard approach is to partition the domain into subregions and employ piecewise linear or constant approximations. This strategy forms the foundation of the variational FEM [Bathe 2006], where nodal vectors are derived by

element-wise integrating and node-wise lumping. In other words, each nodal quantity represents a local integration over its surrounding region in the discrete setting. Analogously, the inner product of two vectors, e.g., $\boldsymbol{u}(i)^\top \boldsymbol{b}$ in Eq. (7), can be interpreted as the discrete integral of the product of two continuous fields, i.e., $u_i(\boldsymbol{x})$ and $b(\boldsymbol{x})$, over the domain:

$$\boldsymbol{u}(i)^\top \boldsymbol{b} = \sum_k u_k(i)^\top b_k = \sum_k |\Omega_k| \varphi(\boldsymbol{x}_k) = \int_\Omega \varphi(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}, \tag{8}$$

where $|\Omega_k|$ represents the area or the volume of the k-th node, and the nodal quantities of $\boldsymbol{u}(i)$ and $\boldsymbol{b}$ are $u_k(i)$ and b_k. $\varphi(\boldsymbol{x})$ stands for the *density* of the product $u_i(\boldsymbol{x})b(\boldsymbol{x})$. It can be evaluated at the k-th node as:

$$\varphi(\boldsymbol{x}_k) = \frac{1}{|\Omega_k|} u_k(i)^\top b_k. \tag{9}$$

Eq. (9) defines φ only at nodes, yet quadrature points do not necessarily fall exactly on a node. We therefore need a continuum extension of the inner product. In the spirit of the Riesz representation theorem [Brenner and Scott 2008], we define the continuous density as the interpolant of the nodal values in Eq. (9):

$$\varphi(\boldsymbol{x}) = \sum_k N_k(\boldsymbol{x})\, \varphi(\boldsymbol{x}_k), \qquad |\Omega_k| := \int_\Omega N_k(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}, \tag{10}$$

with N_k the barycentric shape functions of the mesh and $|\Omega_k|$ the lumped nodal volume (or area). This choice reproduces the inner product exactly: by linearity, $\int_\Omega \varphi\, \mathrm{d}\boldsymbol{x} = \sum_k \varphi(\boldsymbol{x}_k) \int_\Omega N_k\, \mathrm{d}\boldsymbol{x} = \sum_k |\Omega_k| \varphi(\boldsymbol{x}_k)$, which is the last equality in Eq. (8). Evaluating φ at a non-nodal quadrature point then reduces to barycentric interpolation over the enclosing element.

Without prior knowledge of the integrand $\varphi(\boldsymbol{x})$, an even and dense domain partition for creating Ω_k is an obvious option. However, in our case, this integrand is *not* arbitrary. Instead, it possesses specific mathematical properties that can be exploited to simplify the integration process significantly.

Fig. 2 visualizes $\boldsymbol{u}(i)$ as a height field of the PD system for a square elastic body under varying stiffness. It represents the field $u_i(\boldsymbol{x})$ that peaks at node i with $u_i(\boldsymbol{x}_i) = 1$, decays smoothly with distance, and flattens in remote regions. While $b(\boldsymbol{x})$ is not explicitly known, it comprises per-constraint responses $\sum_j w_j \boldsymbol{S}_j^\top \boldsymbol{A}_j^\top \boldsymbol{B}_j \boldsymbol{p}_j$ damped by the inertial term $\frac{\boldsymbol{M}}{\Delta t^2}\boldsymbol{q}^*$ according to Eq. (3). Assuming elasticity stiffness remains relatively uniform, it is reasonable to consider $b(\boldsymbol{x})$ a smooth function. Leveraging these spatial priors of the integrand, we design a multi-level quadrature rule to accelerate the computation of $\boldsymbol{u}(i)^\top \boldsymbol{b}$ using different integration strategies for the near, middle, and far fields relative to node i. This distance-graded strategy is philosophically analogous to the Barnes-Hut algorithm [Barnes and Hut 1986], which likewise resolves nearby contributions individually while summarizing distant clusters by single representatives. This quadrature scheme is agnostic of $b(\boldsymbol{x})$. As long as $b(\boldsymbol{x})$ does not significantly alter the *polynomiality* of $u_i(\boldsymbol{x})b(\boldsymbol{x})$, the integral $\int \varphi(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$ can be accurately obtained with (very) sparse samples. Normally, the so-called polynomiality refers to how well $\varphi(\boldsymbol{x})$ is approximated by a polynomial function of a certain degree. Because any smooth function can be Taylor expanded, we use the polynomiality of $\varphi(\boldsymbol{x})$ to indicate the order of its approximating polynomial.

5.1 Near-field quadrature

As illustrated in Fig. 2, the region immediately surrounding node i, denoted as $\Omega_{near}(i)$, constitutes the most critical domain for the integral $\int \varphi(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$. Although smooth, $\varphi(\boldsymbol{x})$ exhibits high-frequency components in the near field, making standard GL quadrature suboptimal unless high-order rules are employed. In 3D settings, the computational cost escalates cubically with the quadrature order. For instance, increasing from a 3-point to a 4-point GL rule raises the required samples from 27 to 64. This complexity suggests that polynomials are not the most efficient surrogates for $\varphi(\boldsymbol{x})$ in $\Omega_{near}(i)$. Instead, we propose an exponential surrogate, which achieves high integration accuracy with significantly fewer quadrature points.

5.1.1 Local exponential decay. It is known that the PD global system matrix of Eq. (3) closely resembles a shifted Laplacian operator $\mathcal{T}$ in the form of [Brenner and Scott 2008]:

$$\boldsymbol{H} \cong \mathcal{T} \;=\; \alpha \mathcal{I} - \beta \Delta, \tag{11}$$

where $\alpha, \beta > 0$; $\mathcal{I}$ is the identity operator; and Δ is the Laplacian operator. Recalling that $\boldsymbol{u}(i) = [1, -\boldsymbol{h}_i^\top \boldsymbol{H}_i^{-1}]^\top$, it is straightforward to verify that $\boldsymbol{u}(i)$ satisfies the following equilibrium:

$$\boldsymbol{H}\boldsymbol{u}(i) = \begin{bmatrix} H_i & \boldsymbol{h}_i^\top \\ \boldsymbol{h}_i & \boldsymbol{H}_i \end{bmatrix} \begin{bmatrix} 1 \\ -\boldsymbol{H}_i^{-1}\boldsymbol{h}_i \end{bmatrix} = \boldsymbol{l}(i) = \begin{bmatrix} l_i \\ \boldsymbol{0} \end{bmatrix}. \tag{12}$$

Here, $\boldsymbol{l}(i) \in \mathbb{R}^N$ is a loading vector with a single nonzero entry l_i at node i. This suggests that $\boldsymbol{u}(i)$ can be viewed as the impulse response of the discrete system characterized by $\boldsymbol{H}$. Given the similarity of $\boldsymbol{H}$ and $\mathcal{T}$, we infer the pattern of $\boldsymbol{u}(i)$ through examining the behavior of $\mathcal{T}$ under the external impulse applied at $\boldsymbol{x}_i$:

$$\boldsymbol{H}\boldsymbol{u}(i) = \boldsymbol{l}(i) \quad \cong \quad \mathcal{T}\tilde{u}_i(\boldsymbol{x}) = \delta(\boldsymbol{x} - \boldsymbol{x}_i), \tag{13}$$

where δ is the Dirac delta function. In 3D free space, $\tilde{u}_i$ admits a closed-form Green's function [Han and Yin 2026]:

$$\tilde{u}_i(\boldsymbol{x}) = G(\|\boldsymbol{x} - \boldsymbol{x}_i\|) = G(r) = \frac{1}{4\pi\beta r} e^{-r/\lambda}, \tag{14}$$

where $\lambda = \sqrt{\frac{\beta}{\alpha}}$, and $r = \|\boldsymbol{x} - \boldsymbol{x}_i\|$ is the distance from the impulse position. $G(r)$ is also known as the Yukawa kernel, which prescribes an exponential decay of the impulse response. For a sufficiently small near field $|\Omega_{near}(i)| \to 0$, we have $u_i(\boldsymbol{x}) \to \tilde{u}_i(\boldsymbol{x})$.

5.1.2 Near-field integration via Gaussian fit. If u_i fits the Yukawa kernel well per Eq. (14) for $\boldsymbol{x} \in \Omega_{near}$, it is then reasonable to assume that φ can be better approximated by a Gaussian than by a polynomial. For each node i, we define its near field $\Omega_{near}(i)$ as an axis-aligned rectangular region that spans $\left[x_{i,x} - H_{i,x}^{near-},\ x_{i,x} + H_{i,x}^{near+}\right] \times \left[x_{i,y} - H_{i,y}^{near-},\ x_{i,y} + H_{i,y}^{near+}\right] \times \left[x_{i,z} - H_{i,z}^{near-},\ x_{i,z} + H_{i,z}^{near+}\right]$. We defer the detailed discussion of the partitioning process for the near-field domain $\Omega_{near}(i)$ to Sec. 5.4. The inner product $\boldsymbol{u}(i)^\top \boldsymbol{b}$ within $\Omega_{near}(i)$ is then estimated via a definite integral of a surrogate function $\tilde{\varphi}_{near}$ such that $\int_{\Omega_{near}(i)} \varphi \mathrm{d}\boldsymbol{x} \approx \int_{\Omega_{near}(i)} \tilde{\varphi}_{near} \mathrm{d}\boldsymbol{x}$, where the surrogate is an anisotropic 3D Gaussian G_i, defined as:

$$\tilde{\varphi}_{near}(\boldsymbol{x}) = G_i(\boldsymbol{x}) = A_i \prod_{\alpha \in \{x,y,z\}} e^{-(x_\alpha - x_{i,\alpha})^2 / \sigma_{i,\alpha}^2}. \tag{15}$$

The near-field integration has a closed-form approximation of:

$$\int_{\Omega_{near}(i)} \varphi(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \int_{\Omega_{near}(i)} \tilde{\varphi}_{near}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = \int_{\Omega_{near}(i)} G_i(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$$
$$= \frac{A_i\,\pi^{\frac{3}{2}}}{8} \prod_{\alpha\in\{x,y,z\}} \sigma_{i,\alpha} \left[\mathrm{erf}\left(\frac{H^{near+}_{i,\alpha}}{\sigma_{i,\alpha}}\right) + \mathrm{erf}\left(\frac{H^{near-}_{i,\alpha}}{\sigma_{i,\alpha}}\right)\right], \quad (16)$$

where $\mathrm{erf}(x)$ is the error function, whose evaluation is given by $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\mathrm{d}t$. In other words, we need to compute 4 kernel parameters $(A_i, \sigma_{i,x}, \sigma_{i,y}, \sigma_{i,z})$, and the near-field inner product can be analytically obtained without iterating all the nodes in Ω_{near}.

These unknown Gaussian parameters are determined at runtime via an augmented least-squares fitting:

$$\min_{A_i,\sigma_{i,x},\sigma_{i,y},\sigma_{i,z}} \sum_{\boldsymbol{x}_s\in\mathcal{S}_i} \left(G_i(\boldsymbol{x}_s) - \varphi(\boldsymbol{x}_s)\right)^2 + \mu \sum_{\boldsymbol{x}_f\in\mathcal{F}_i} G_i^2(\boldsymbol{x}_f). \quad (17)$$

The target function $\varphi(\boldsymbol{x}_s)$ is queried via Eq. (9) at a set of sample nodes $\mathcal{S}_i$. This set always includes the current node i, while the remaining nodes are selected within Ω_{near} using Farthest Point Sampling (FPS) [Gonzalez 1985] with the geodesic distance metric. In our implementation, we set $|\mathcal{S}_i| = 8$ to ensure the fitting problem in Eq. (17) is slightly over-constrained.

The second term in Eq. (17) serves as a smoothing penalty. As discussed later in Sec. 5.2, our near-field integration is actually superposed with a GL quadrature-based middle-field integration. We set $\mu = 100$, i.e., two orders of magnitude larger than the maximum value of $|u(i)|$ to ensure that the residual $\varphi(\boldsymbol{x}) - G_i(\boldsymbol{x})$ remains smooth over Ω_{middle}, which effectively improves the accuracy of low-order GL integration (e.g., see Fig. 3, right column). This penalty strongly damps the Gaussian surrogate near the boundary between Ω_{near} and Ω_{middle}. Hence, $\mathcal{F}_i$ consists of six nodes near the centers of the six faces of Ω_{near}.

Although the least-squares fitting in Eq. (17) appears complex, it only requires solving a 4-DOF linear system for $(A_i, \sigma_{i,x}, \sigma_{i,y}, \sigma_{i,z})$ by sampling a small number of function values at nodes in $\mathcal{S}_i$ and $\mathcal{F}_i$. We implement a hand-coded Cholesky factorization, which requires only a few dozen floating-point operations for a 4×4 system. In contrast, Ω_{near} can house hundreds or even thousands of nodes for high-resolution models. A brute-force inner product within this domain becomes several times more expensive than our fitting-based approach.

5.2 Middle-field quadrature

The near-field Gaussian integration effectively tackles the most salient geometric features near the peak of node i. Once this peak is taken care of, the remaining influence of φ, as observed in Fig. 2, becomes smooth enough to be well-suited for a *single* low-degree polynomial approximation. Consequently, instead of subdividing $\Omega_{middle} \setminus \Omega_{near}$ into multiple box-shaped domains for piecewise integration [Guo et al. 2026a], we employ a more efficient strategy that performs sparse quadrature over the middle field that encompasses the near field (i.e., $\Omega_{near} \subset \Omega_{middle}$), leveraging the GL rule discussed in Sec. 3.2. This avoids unnecessary computational overhead of managing multiple rectangular sub-regions while maintaining high integration accuracy.

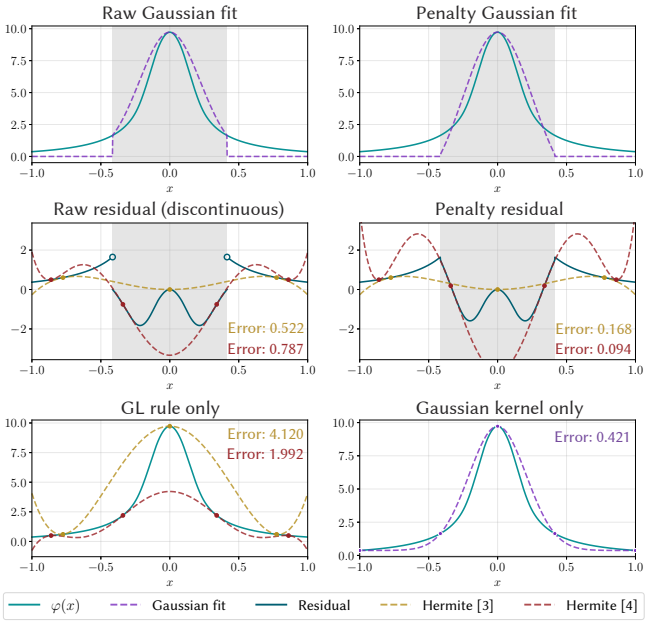

Fig. 3. We show a representative 1D example of the proposed middle-field integration using the stacking GL rule. The target function φ, peaking at $x = 0$, is the analytic 1D solution to Eq. (14), regularized to avoid singularity. The middle-field domain Ω_{middle} is normalized to $[-1, 1]$, where the gray region denotes Ω_{near}. Within Ω_{near}, φ is approximated by a Gaussian G_i via the least-squares fitting in Eq. (17). Therefore, G_i vanishes outside Ω_{near}. The integral of the residual, i.e., the difference between φ and G_i over Ω_{middle}, is then estimated using low-order GL quadrature. The middle row plots the residual $\varphi - G_i$ across Ω_{middle}. Notably, the penalty term in Eq. (17) effectively smoothens this residual. We also visualize the polynomial surrogates corresponding to the 3-point and 4-point GL rules. Compared to plain GL quadrature or Gaussian-only fitting, as shown in the bottom row, the stacking GL rule yields lower integration error.

We refer to our middle-field quadrature as the *stacking GL rule* because Ω_{middle} contains Ω_{near}, the contribution of which has already been evaluated via Eq. (16). Specifically, Ω_{middle} is an axis-aligned box with dimensions of $\left[x_{i,x} - H^{middle-}_{i,x},\ x_{i,x} + H^{middle+}_{i,x}\right] \times \left[x_{i,y} - H^{middle-}_{i,y},\ x_{i,y} + H^{middle+}_{i,y}\right] \times \left[x_{i,z} - H^{middle-}_{i,z},\ x_{i,z} + H^{middle+}_{i,z}\right]$. It fully encloses Ω_{near}. To avoid overcounting, the target function within Ω_{middle} is defined as the piecewise function:

$$\tilde{\varphi}_{middle}(\boldsymbol{x}) = \begin{cases} \varphi(\boldsymbol{x}) - G_i(\boldsymbol{x}), & \boldsymbol{x}\in\Omega_{near}, \\ \varphi(\boldsymbol{x}), & \boldsymbol{x}\in\Omega_{middle}\setminus\Omega_{near}. \end{cases} \quad (18)$$

Specifically, this middle-field quadrature approximates the integral over Ω_{middle} by excluding the contributions already accounted for by the near-field Gaussian surrogate G_i.

The incorporation of the penalty term $\mu\sum G_i^2(\boldsymbol{x}_f)$ in Eq. (17) now becomes evident. Assuming φ is smooth, the term $\varphi(\boldsymbol{x}) - G_i(\boldsymbol{x})$ remains smooth within Ω_{near}. Consequently, any potential discontinuity in $\tilde{\varphi}_{middle}(\boldsymbol{x})$ must occur at the boundary $\partial\Omega_{near}$. To improve the accuracy of the middle-field low-order GL quadrature,

we minimize the cross-domain jump:

$$\int_{\partial\Omega_{near}} \big((\varphi(\boldsymbol{x}) - G_i(\boldsymbol{x})) - \varphi(\boldsymbol{x})\big)^2 \,\mathrm{d}\boldsymbol{x} = \int_{\partial\Omega_{near}} G_i^2(\boldsymbol{x})\mathrm{d}\boldsymbol{x}. \quad (19)$$

A representative 1D example is illustrated in Fig. 3, highlighting the effectiveness of our stacking quadrature strategy for kernel-shaped functions. Since the peak region is accurately captured by the Gaussian fit, a low-order GL rule is sufficient to estimate the integral of the residual function, $\varphi - G_i$. This approach achieves significantly higher integration accuracy than vanilla GL quadrature while using far fewer quadrature samples.

The contribution from $\Omega_{middle}(i)$ to $\boldsymbol{u}(i)^\top \boldsymbol{b}$ can then be approximated via applying the GL quadrature rule (i.e., Eq. (4)) within Ω_{middle}:

$$\int_{\Omega_{middle}} \tilde{\varphi}_{middle}\mathrm{d}\boldsymbol{x} \approx J \sum_{k_x=1}^{n_x}\sum_{k_y=1}^{n_y}\sum_{k_z=1}^{n_z} w_{k_x} w_{k_y} w_{k_z}\, \tilde{\varphi}_{middle}\big(\boldsymbol{\xi}_{k_x k_y k_z}\big), \quad (20)$$

where $J = \frac{1}{8}(H_{i,x}^{middle-} + H_{i,x}^{middle+})(H_{i,y}^{middle-} + H_{i,y}^{middle+})(H_{i,z}^{middle-} + H_{i,z}^{middle+})$. The function $\tilde{\varphi}_{middle}$ is queried piecewise according to Eq. (18), depending on the positions of the GL quadrature samples. In practice, we find that setting $n_x = n_y = n_z = 3$ provides sufficient accuracy. Notably, the weight coefficients and sample positions for the GL quadrature are analytically available. Therefore, our method requires no additional runtime computation beyond querying the target function values at these sample points.

The positions of GL quadrature points $\boldsymbol{\xi}_{k_x k_y k_z}$ are determined by the tensor-product rule on Ω_{middle} and therefore do not generally coincide with mesh nodes. We thus sample the interpolant field in Eq. (10) for these off-node quadrature points. As $\Omega_{middle}(i)$ is fixed throughout the simulation, both the enclosing element and barycentric weights can be determined at precomputation time.

5.3 Far-field quadrature

The Gaussian kernel fit and GL quadrature together accurately capture the contribution in the neighborhood around node i. What remains is an even smoother residual over the rest of the geometry $\Omega_{far}(i) = \Omega \setminus \Omega_{middle}(i)$. $\Omega_{far}(i)$ is often irregular due to the carved-out rectangular box of Ω_{middle}. This geometric irregularity precludes the use of rules that require a simple domain, such as GL. We therefore resort to a clustering algorithm that places no restrictions on the domain geometry to approximate $\int_{\Omega_{far}} \varphi(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$, as each cluster, together with its data points and centroid, induces a quadrature scheme.

Clusters are formed during precomputation. We first partition $\Omega_{far}(i)$ into maximal face-connected components $\{\Omega_{far}^{(\ell)}(i)\}_{\ell=1}^{L}$ via breadth-first search, and then form clusters independently within each component. This ensures that every centroid lies inside its associated cluster, which would not hold were a cluster allowed to span disconnected components. Within each connected component $\Omega_{far}^{(\ell)}(i)$, K_ℓ clusters are formed by applying the weighted k-means++ algorithm [Arthur and Vassilvitskii 2007]. The data points are the nodes in $\Omega_{far}^{(\ell)}(i)$, and the weights are the absolute values of the corresponding entries of $\boldsymbol{u}(i)$. Seeding picks the highest-weight node as the first centroid; and each subsequent centroid is sampled from the remaining nodes with probability proportional to the product of the node's weight and the squared geodesic distance to its nearest existing centroid. Lloyd iterations then alternate between assignment and weighted-mean updates until convergence.

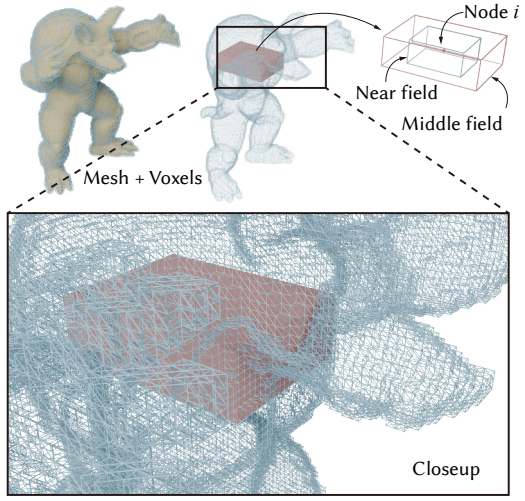

Fig. 4. For a given node i, a seed voxel is expanded into a maximal rectangular volume to form the middle-field Ω_{middle}, capturing up to 90% of the cumulative influence $\int_\Omega \omega_i(\xi)\mathrm{d}\xi$. The near-field Ω_{near} is defined as the nested sub-domain capturing 70% of this influence, while the far-field Ω_{far} (not explicitly boxed) comprises the remaining irregular domain of residual voxels. The closeup illustrates the alignment of the cuboid expansion with the underlying voxelized mesh structure.

Based on the resulting clusters, we define a quadrature rule using the Lloyd-converged centroids. A natural choice of quadrature point is the node $\boldsymbol{c}_k^{(\ell)}$ closest to the centroid of $C_k^{(\ell)}$, the k-th cluster in $\Omega_{far}^{(\ell)}(i)$. The contribution from Ω_{far} is then approximated as

$$\int_{\Omega_{far}} \varphi(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \sum_{\ell=1}^{L}\sum_{k=1}^{K_\ell} \left|C_k^{(\ell)}\right| \varphi\left(\boldsymbol{c}_k^{(\ell)}\right), \quad (21)$$

where $\left|C_k^{(\ell)}\right|$ denotes the volume of cluster $C_k^{(\ell)}$, computed as the sum of the lumped volumes of its constituent nodes. We find that a total of $\sum_\ell K_\ell = 8$ clusters, allocated across connected components in proportion to their volumes, are in general sufficient.

5.4 Integration domain partition

The near, middle, and far fields for each node are defined using a series of nested cuboid domains. First, the axis-aligned bounding box (AABB) of the input mesh is partitioned into an $H_x \times H_y \times H_z$ voxel grid with a resolution of $1/h$. Voxels intersecting the mesh volume or surface are assigned a value of `flag` = 1 (Alg. 1, lines 2–4), while all others are set to 0. Each node i is associated with a specific voxel v (where `flag` = 1), which serves as the seed for a cuboid domain. Following [Guo et al. 2026a], the *influence magnitude* from node j to node i is defined as $\omega_i(\boldsymbol{x}_j) = |u(i,j)|$, where $u(i,j)$ denotes the j-th entry of $\boldsymbol{u}(i)$. For any point within a volumetric element (such as a tetrahedron or hexahedron), this influence is

evaluated via barycentric interpolation of the discretized ω_i values. The cumulative influence over the entire domain Ω is given by the integral $\int_\Omega \omega_i(\xi)\mathrm{d}\xi$. This seed is greedily expanded along its axes to form a maximal rectangular volume of voxels that all maintain `flag` = 1 (Alg. 1, lines 7–13), but not larger than 90% of the cumulative influence (Fig. 4, upper right, pink box). This expanded volume is designated as Ω_{middle}, and it contains the near-field region Ω_{near}, which is defined later. Accordingly, the near-field Ω_{near} is identified as the region within Ω_{middle} that captures a specific fraction of the cumulative influence, satisfying $\int_{\Omega_{near}} \omega_i(\xi)\mathrm{d}\xi = 0.7 \int_\Omega \omega_i(\xi)\mathrm{d}\xi$ (Fig. 4, upper right, blue box). Finally, Ω_{far} is defined as the remaining domain (Alg. 1, line 16). Ω_{far} is highly irregular and is composed of the residual voxels.

Algorithm 1: Field domains generation for node i

Input: Input mesh $\mathcal{M}$, node i, resolution $1/h$, influence function ω_i
Output: Domains Ω_{near}, Ω_{middle}, Ω_{far}

```
// Stage 1: Voxelization and domain initialization
V ← Partition(AABB(M), H_x × H_y × H_z);
// Define the entire valid domain
Ω ← {v ∈ V | v ∩ M ≠ ∅};
foreach v ∈ V do
    v.flag ← (v ∈ Ω) ? 1 : 0;
// Stage 2: Middle-field via greedy expansion
v_seed ← {v ∈ Ω | i ∈ v};
Ω_middle ← {v_seed};
while ∃ axis expansion possible for Ω_middle do
    a* ← arg max_{a∈{x,y,z}} Area(CrossSection(Ω_middle, a));
    foreach d ∈ {+, −} along a* do
        S_cand ← adjacent slice of Ω_middle in direction d;
        if ∀v ∈ S_cand, v.flag = 1 then
            Ω_middle ← Ω_middle ∪ S_cand;
            break;
// Stage 3: Near-field constraint
I_total ← ∫_Ω ω_i(ξ)dξ;
Identify Ω_near ⊆ Ω_middle such that ∫_{Ω_near} ω_i(ξ)dξ = 0.7 I_total;
// Stage 4: Far-field residual
Ω_far ← Ω \ Ω_middle;
return Ω_near, Ω_middle, Ω_far;
```

6 Global solve with collisions

The discussion so far assumes the constraint set of the PD system remains unchanged during the simulation. However, this assumption no longer holds when collisions and contacts are present. Consequently, the proposed quadrature rule cannot be applied directly. To address this, we propose a novel nested preconditioning algorithm. This algorithm utilizes our collision-free second-order Jacobi procedure of Eq. (7) as a matrix-free preconditioner for Conjugate Gradient (CG) to solve the global step with a varying $\boldsymbol{H}$.

At each time step, we warm-start the system by assuming all contacts from the previous step are resolved. While typical warm-start strategies advance the system using an explicit or semi-implicit step [Wang and Yang 2016], our second-order Jacobi method is efficient and requires less than one millisecond even for high-resolution models. We therefore utilize a full collision-free solve $\boldsymbol{Hq} = \boldsymbol{b}$ as a more effective warm-start mechanism. Following collision detection, if active constraints are identified, we solve an updated global system:

$$(\boldsymbol{H} + \Delta\boldsymbol{H})(\boldsymbol{q} + \Delta\boldsymbol{q}) = \boldsymbol{b} + \Delta\boldsymbol{b}, \tag{22}$$

where $\Delta\boldsymbol{H}$ and $\Delta\boldsymbol{b}$ account for the contributions from all detected collisions and contacts. With $\boldsymbol{q}$ obtained at the warm-start stage, we solve for the corrective increment $\Delta\boldsymbol{q}$ to accommodate newly identified constraints:

$$(\boldsymbol{H} + \Delta\boldsymbol{H})\Delta\boldsymbol{q} = \Delta\boldsymbol{b} - \Delta\boldsymbol{H}\boldsymbol{q}. \tag{23}$$

Our method is compatible with various collision detection algorithms. One may employ discrete collision detection (DCD) to project constraints based on penetration depth [Chen et al. 2023], or continuous collision detection (CCD) to resolve constraints when surface primitives are within a specified proximity [Lan et al. 2022; Li et al. 2020].

6.1 Contact constraints and their diagonalization

Within the PD framework, collisions are addressed by introducing unilateral constraints that enforce a non-negative distance between contact pairs. Let $\boldsymbol{a}_c$ and $\boldsymbol{b}_c$ be the contacting points at each colliding primitive for the c-th collision pair, and $\boldsymbol{n}_c$ be a unit vector representing the collision normal. The constraint manifold is simply $d_c = \boldsymbol{n}_c^\top(\boldsymbol{a}_c - \boldsymbol{b}_c) \geq 0$.

Consider a point-triangle contact involving four nodes $\boldsymbol{q}_c = \boldsymbol{S}_c\boldsymbol{q} \in \mathbb{R}^{4\times 3}$, where the first row $\boldsymbol{q}_0$ contains the coordinates of the colliding point and the other three rows $\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3$ denote the triangle vertices. The local problem associated with this constraint boils down to:

$$\arg\min_{p_c \geq \epsilon} \Psi_c, \quad \Psi_c = \frac{w_c}{2}\left(\boldsymbol{\alpha}\boldsymbol{S}_c\boldsymbol{q}\boldsymbol{n}_c - p_c\right)^2, \tag{24}$$

where $\boldsymbol{\alpha} = [1, -\alpha_1, -\alpha_2, -(1-\alpha_1-\alpha_2)] \in \mathbb{R}^{1\times 4}$ is a row vector encoding the barycentric coordinates of the contact point $\boldsymbol{b}_c$ on the triangle. For an edge-edge contact, $\boldsymbol{\alpha} = [\alpha, (1-\alpha), -\beta, -(1-\beta)]$. w_c is the stiffness or weight of the collision constraint, and $\epsilon \geq 0$ is the separation threshold. The optimal auxiliary variable p_c is obtained by projecting the current signed distance onto the feasible set:

$$p_c = \max\left\{\boldsymbol{\alpha}\boldsymbol{S}_c\boldsymbol{q}\boldsymbol{n}_c, \epsilon\right\}. \tag{25}$$

The contribution of this constraint to the global system matrix $\Delta\boldsymbol{H}_c$ and the right-hand side vector $\Delta\boldsymbol{b}_c$ can be explicitly derived as:

$$\Delta\boldsymbol{H}_c = w_c\left(\boldsymbol{S}_c^\top\boldsymbol{\alpha}^\top\boldsymbol{\alpha}\boldsymbol{S}_c\right), \quad \Delta\boldsymbol{b}_c = w_c\left(\boldsymbol{S}_c^\top\boldsymbol{\alpha}^\top p_c\right)\boldsymbol{n}_c^\top. \tag{26}$$

Eq. (26) constructs a 4×4 stencil for each collision constraint, which is then scattered into the global matrix via $\boldsymbol{S}_c$. This inevitably introduces off-diagonal entries to $\Delta\boldsymbol{H}$, resulting in a sparsity pattern less friendly for Jacobi-style solvers. Furthermore, since $\Delta\boldsymbol{H}$ is not guaranteed to be diagonally dominant, the effectiveness of Jacobi preconditioning becomes questionable. These numerical limitations motivate us to adopt an alternative collision projection formulation that ensures $\Delta\boldsymbol{H}$ remains strictly diagonal.

Concretely, the collision penalty is defined based on the squared Euclidean distance between $\boldsymbol{q}_c$ and $\boldsymbol{p}_c$ instead of on penetration

depth or separating distance along the collision normal. The local problem is thus re-formulated as:

$$\arg\min_{\boldsymbol{p}_c} \tilde{\Psi}_c,\ \tilde{\Psi}_c = \frac{w_c}{2}\|\boldsymbol{S}_c\boldsymbol{q} - \boldsymbol{p}_c\|_F^2 \text{ s.t. } \boldsymbol{p}_c \text{ is penetration-free.} \quad (27)$$

In this formulation, the auxiliary variable $\boldsymbol{p}_c \in \mathbb{R}^{4\times 3}$ represents a set of nodal positions that are proximal to $\boldsymbol{S}_c\boldsymbol{q}$ while satisfying the non-penetration condition. For point-triangle or edge-edge contacts, $\boldsymbol{p}_c$ is efficiently obtained by projecting the current configuration $\boldsymbol{S}_c\boldsymbol{q}$ onto the best-fitting plane formed by four participating nodes.

The corresponding $\boldsymbol{A}_j$ and $\boldsymbol{B}_j$ as in Eq. (2) are all identity matrices, and its contribution to the global matrix becomes:

$$\Delta\boldsymbol{H}_c = w_c\left(\boldsymbol{S}_c^\top\boldsymbol{S}_c\right). \quad (28)$$

Since $\boldsymbol{S}_c$ is a selection matrix, $\Delta\boldsymbol{H}_c$ is diagonal.

Explicitly setting $\boldsymbol{A}_j$ and $\boldsymbol{B}_j$ as identity matrices as in Eq. (27) improves the sparsity of the global system. On the downside, converting geometric constraints into point-wise distance penalties does not eliminate the nullspace of the constraint manifold, which results in a slower convergence rate. In contrast, the formulation in Eq. (26) converges significantly faster but generates off-diagonal non-zeros, making the solver less compatible with Jacobi-style parallelism. Our strategy borrows the best from both: we exploit Eq. (27) as a preconditioning operator to solve Eq. (23), where $\Delta\boldsymbol{H}$ is assembled with Eq. (26).

Algorithm 2: Nested preconditioned CG

Input: $\widetilde{\boldsymbol{H}}, \tilde{\boldsymbol{b}}, \mathcal{P}_c, \mathcal{P}_e, k_c, k_e$
Output: $\Delta\boldsymbol{q}$
// $\Delta\boldsymbol{q}$ is initialized as a zero vector
$\Delta\boldsymbol{q}_0 \leftarrow \boldsymbol{0}$;
$\boldsymbol{r}_0 \leftarrow \tilde{\boldsymbol{b}}$;
// $\tilde{\boldsymbol{r}}$ is the preconditioned residual
$\tilde{\boldsymbol{r}}_0 \leftarrow \mathcal{P}_e^{k_e}\left[\mathcal{P}_c^{k_c}(\boldsymbol{r}_0)\right]$;
// $\boldsymbol{s}$ is the preconditioned conjugate vector
$\boldsymbol{s}_0 \leftarrow \tilde{\boldsymbol{r}}_0$;
$k = 0$;
// Convergence check based on raw residual
while $\|\boldsymbol{r}_{k+1}\| > 10^{-4}$ *and* $k \le 2$ **do**
  $\alpha_k \leftarrow \frac{\boldsymbol{r}_k^\top\tilde{\boldsymbol{r}}_k}{\boldsymbol{s}_k^\top\widetilde{\boldsymbol{H}}\boldsymbol{s}_k}$;
  $\Delta\boldsymbol{q}_{k+1} \leftarrow \Delta\boldsymbol{q}_k + \alpha_k\boldsymbol{s}_k$;
  $\boldsymbol{r}_{k+1} \leftarrow \boldsymbol{r}_k - \alpha_k\widetilde{\boldsymbol{H}}\boldsymbol{s}_k$;
  $\tilde{\boldsymbol{r}}_{k+1} \leftarrow \mathcal{P}_e^{k_e}\left[\mathcal{P}_c^{k_c}(\boldsymbol{r}_{k+1})\right]$;
  $\beta_k \leftarrow \frac{\boldsymbol{r}_{k+1}^\top\tilde{\boldsymbol{r}}_{k+1}}{\boldsymbol{r}_k^\top\tilde{\boldsymbol{r}}_k}$;
  $\boldsymbol{s}_{k+1} \leftarrow \tilde{\boldsymbol{r}}_{k+1} + \beta_k\boldsymbol{s}_k$;
  $k \leftarrow k+1$;
return $\Delta\boldsymbol{q}_{k+1}$;

6.2 Nested preconditioning

We now have two effective numerical tools to handle elasticity and contact independently. First, the second-order Jacobi solver with mixed quadrature (Sec. 4) provides a highly efficient framework for collision-free cases. Second, by using a point-wise penalty formulation, we can diagonalize the matrix increments for contact constraints, making the subproblem Jacobi-friendly. The challenge is to integrate these treatments into the contact-aware PD system. Our goal is to maximize parallelism while maintaining the fast convergence associated with the original geometric formulation, i.e., using Eq. (26).

Our baseline solution to Eq. (23) is CG since $\Delta\boldsymbol{H}$ varies under different collision configurations, and precomputed quadratures are not directly helpful. As mentioned, we employ the second-order Jacobi and diagonalized contact as two *preconditioning operators* during CG iterations. It is known that the preconditioned CG can be more effective than regular CG if the preconditioning matrix $\boldsymbol{P}$ captures the essential spectral feature of the system matrix. It converts Eq. (23) to:

$$\widetilde{\boldsymbol{H}}\Delta\boldsymbol{q} = \widetilde{\boldsymbol{b}} \rightarrow \boldsymbol{P}\widetilde{\boldsymbol{H}}\Delta\boldsymbol{q} = \boldsymbol{P}\widetilde{\boldsymbol{b}}, \quad (29)$$

where $\widetilde{\boldsymbol{H}} = \boldsymbol{H} + \Delta\boldsymbol{H}$, and $\widetilde{\boldsymbol{b}} = \Delta\boldsymbol{b} - \Delta\boldsymbol{H}\boldsymbol{q}$. A good preconditioner yields a condition number $\kappa(\boldsymbol{P}\boldsymbol{H})$ much lower than $\kappa(\boldsymbol{H})$, thereby accelerating the convergence of CG iterations, which follows the rate of $\rho = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$. Designing a preconditioner that is both effective and computationally efficient is non-trivial. In graphics, a common practice is to use the inverse of the diagonal or diagonal blocks of the system matrix for GPU-based CG [Guo et al. 2024; Huang et al. 2024], or incomplete Cholesky factorization for CPU-based implementations [Baraff and Witkin 1998].

We precondition Eq. (23) using two operators, namely $\mathcal{P}_c$ and $\mathcal{P}_e$. The contact preconditioning operator $\mathcal{P}_c$ is defined as:

$$\mathcal{P}_c(\tilde{\boldsymbol{q}}) = \arg\min_{\boldsymbol{q}} \frac{1}{2\Delta t^2}\|\boldsymbol{M}^{\frac{1}{2}}(\boldsymbol{q} - \tilde{\boldsymbol{q}})\|_F^2 + \tilde{\Psi}_c(\boldsymbol{q}). \quad (30)$$

In this formulation, $\mathcal{P}_c$ takes $\tilde{\boldsymbol{q}}$ as input and outputs $\boldsymbol{q}$ by solving a simplified variational problem that includes only the inertia term and the diagonalized collision potential $\tilde{\Psi}_c$ using PD. Because the global matrix of Eq. (30) is diagonal, it can be solved by a single Jacobi iteration. In other words, $\mathcal{P}_c$ mimics the collision response of the system without considering the elasticity. While $\tilde{\Psi}_c$ is less accurate compared with using Ψ_c, its fully diagonal Hessian eases the computation. In our implementation, we apply k_c LG iterations, denoted as $\mathcal{P}_c^{k_c}(\tilde{\boldsymbol{q}})$. After each LG iteration, we update $\boldsymbol{p}_c$ for each contact constraint to adjust the best-fitting penetration-free position for each contact node.

The second operator, $\mathcal{P}_e^{k_e}$, is embodied by the collision-free PD solve using second-order Jacobi iterations and mixed quadrature, executed for k_e LG iterations. Notably, both $\mathcal{P}_c$ and $\mathcal{P}_e$ are defined as nonlinear procedures rather than explicit matrices. However, like a standard matrix-based preconditioner $\boldsymbol{P}$, they map an input vector to a filtered output vector. Our final preconditioning operator for Eq. (23) is defined as the composition of these two steps:

$$\mathcal{P}(\cdot) \triangleq \mathcal{P}_e^{k_e}\left[\mathcal{P}_c^{k_c}(\cdot)\right]. \quad (31)$$

This operator replaces the traditional matrix multiplication $\boldsymbol{P}\boldsymbol{r}$ used to modify residual and conjugate vectors in standard CG. The pseudo-code for our nested preconditioning scheme is provided in Alg. 2. In our implementation, the PCG does not use more than three iterations, and we have $k_c = 3$ and $k_e = 1$.

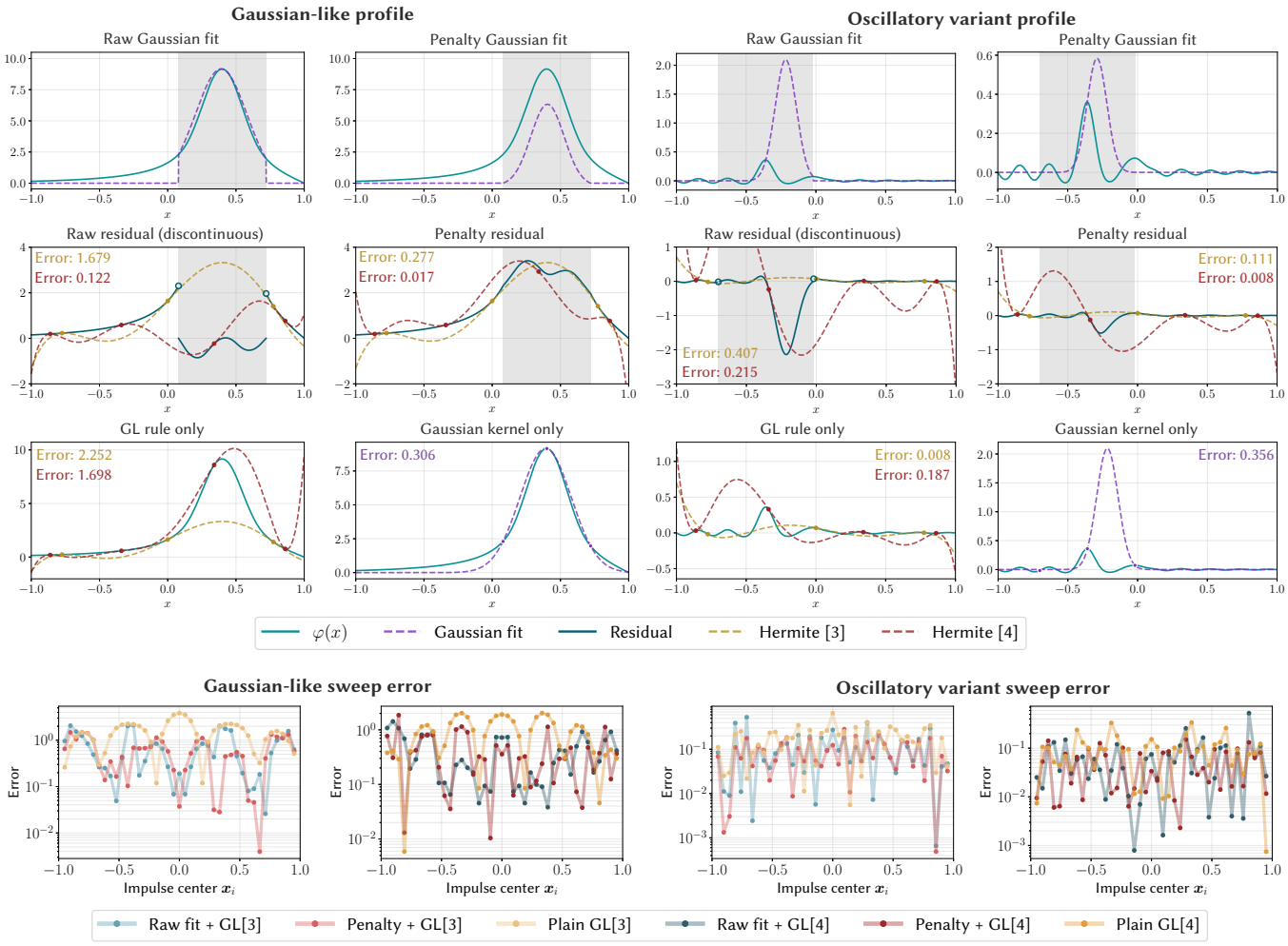

Fig. 5. We evaluate the integration accuracy of our stacking GL quadrature rule on two representative 1D integrands: a Gaussian-like regularized solution of Eq. (14) (left half) and an oscillatory variant (right half). Dashed lines visualize what the GL rule effectively integrates. Gaussian fitting domain for stacking GL rule is shaded, and absolute fitting error is indicated for each rule. The representative example for both cases (top) demonstrates the failure cases of other immediate alternative integration strategies: plain GL incurs large error at the spike; stacking GL rule, without boundary penalty, produces a jump at the fit boundary that contaminates the subsequent GL stage; and the Gaussian kernel alone cannot resolve sign changes. We further sweep the impulse center across the domain and plot the quadrature error for all strategies (bottom). Plain GL rule is highly volatile and sensitive to the target function; and stacking GL rule with penalty gives the best accuracy on both integrands.

7 Experimental results

We implemented the proposed framework on a desktop computer with an AMD Ryzen 9 9950X CPU and an NVIDIA GeForce RTX 4090 GPU. The pre-computation of $\boldsymbol{u}(i)$ is performed on the CPU using Eigen and MKL, requiring less than 10 minutes in general; while the simulation is executed on the GPU. Tab. 1 summarizes the experimental setup and timing statistics for the examples.

Although the denominator $\boldsymbol{u}(i)^\top \boldsymbol{H}\boldsymbol{u}(i)$ of Eq. (7) can be pre-computed exactly, in all the experiments below we evaluate it at runtime with the same multi-level quadrature rule used for the numerator, so that the two estimates carry a consistent integration error that is important for the convergence of our method.

7.1 Integration accuracy

We first evaluate the effectiveness of our integration strategy against the reference function from Eq. (14), regularized through smoothing by a sharp hat function so that no singularities exist. We also evaluate against a variant where the $G(r)$ is multiplied by a sinusoidal term, which preserves the envelope while introducing sign changes. As Eq. (14) only admits an analytic solution in free space, we apply the method of images for an approximation [Evans 2010]. As shown in Fig. 5, plain GL rules fail to resolve the exponential spike, leading to high sensitivity to the peak position, and high integration errors; and stacking GL rule, without penalty on Gaussian fitting domain boundary, yields large jumps on the boundary, which harms the accuracy of subsequent application of GL rule on the fitting residual.

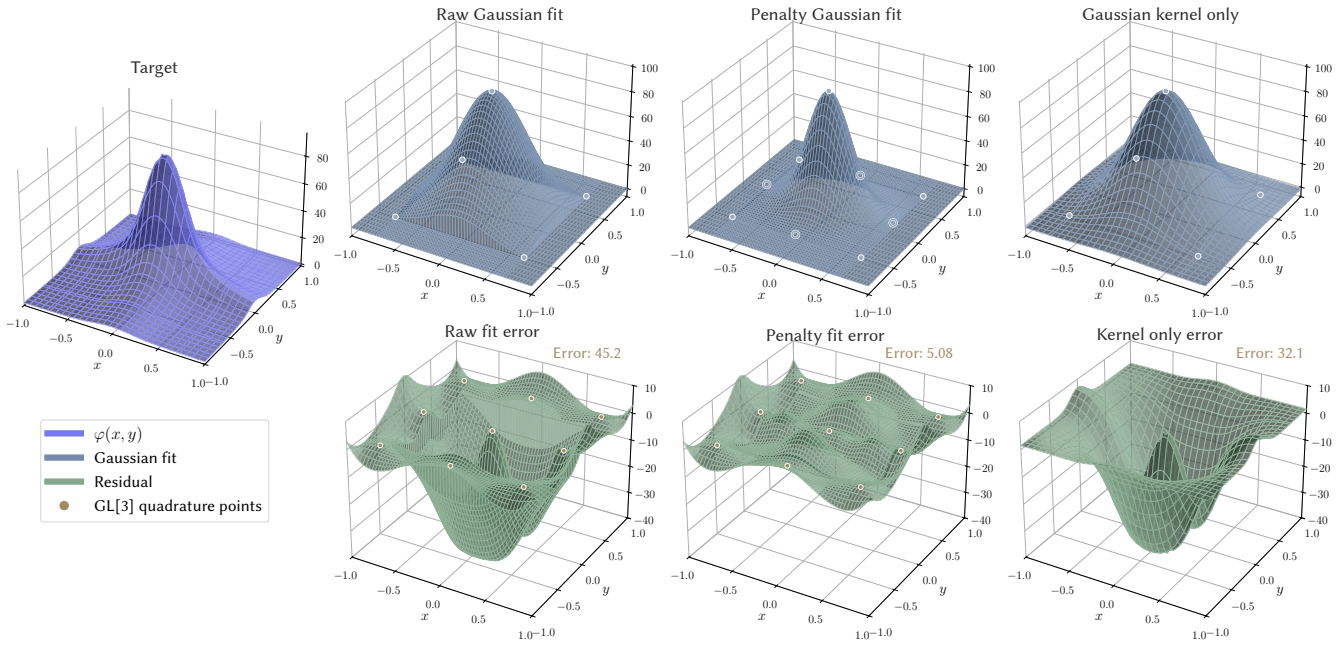

Fig. 6. We perform the same accuracy evaluations in 2D. The target $\varphi(x, y)$, as tensor product of the 1D target in Fig. 5, is approximated by three strategies: stacking GL rule with and without boundary penalty; and a single Gaussian kernel. Each fitted surface is shown in the top row; and the quadrature residue, demonstrated via the difference between the target and the fitted Gaussian and the effective integrand of GL rule if applicable, is shown in the bottom row. Gaussian fitting sample points and GL rule quadrature points are highlighted with filled circles, and boundary-penalty constraint points are indicated by ring markers. Stacking GL rule with penalty distributes the fitting error evenly over the domain, resulting in less quadrature error compared to the other two strategies.

We then conduct the same comparison in the 2-dimensional scenario. From Fig. 6, the tensor-product structure of the target kernel produces a sharper decay compared to the 1-dimensional case. Here, plain Gaussian fitting and stacking GL rule suffer from the same issue: without constraint from the penalty, the Gaussian kernel fitted from a handful of samples captures the exponential decay away from the center poorly. The residual thus preserves the exponential decay structure, which GL rule cannot resolve. The penalty formulation in our quadrature scheme ensures a smoother residual and thus less quadrature error.

7.2 Ablation study

Near-field Gaussian fit vs GL quadrature. The near-field is the most critical domain for the application of the Gaussian fit. We conducted an ablation study comparing the near-field Gaussian fit against Gauss-Legendre (GL) quadrature. When a sharp external force is applied, $\varphi(\boldsymbol{x})$ exhibits high-frequency components in the near field, causing standard GL quadrature to become a poor approximation for $\int_{\Omega_{near}} \varphi(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$. Throughout this subsection, "residual" refers to the relative residual of the linear solve, $\|\boldsymbol{e}^{[l]}\|_2/\|\boldsymbol{e}^{[0]}\|_2$, where $\boldsymbol{e}^{[l]} = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{q}^{[l]}$ is the residual at the end of the l-th iteration. As illustrated in Fig. 7, our Gaussian fit demonstrates significantly higher efficiency compared to the GL quadrature (4-point rule). For instance, at Frame 100, where the armadillo undergoes extreme deformation due to dragging, the Gaussian fit converges in only

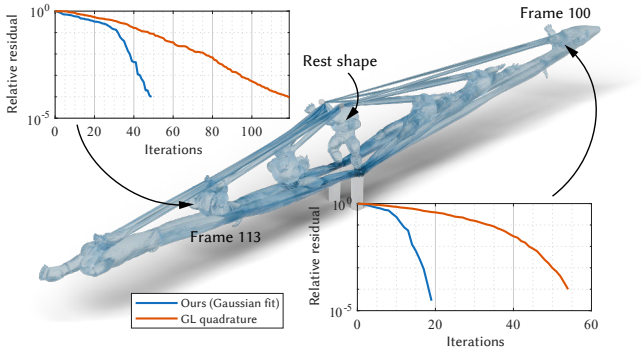

Fig. 7. In the near field, where $\varphi(\boldsymbol{x})$ exhibits high-frequency components, the Gaussian fit (blue) provides a superior approximation compared to the standard GL quadrature (orange), which suffers from reduced accuracy in this critical domain.

37 ms, whereas GL quadrature requires 127 ms. This performance gap becomes even more pronounced at Frame 113 during the rapid "bounce-back" phase; here, the Gaussian fit reaches convergence in 77 ms, while GL quadrature takes 293 ms, nearly four times longer.

Near-middle stacking vs non-stacking. An alternative strategy involves expanding outward from the six faces of Ω_{near} and performing quadrature over these non-overlapping boxed sub-regions using

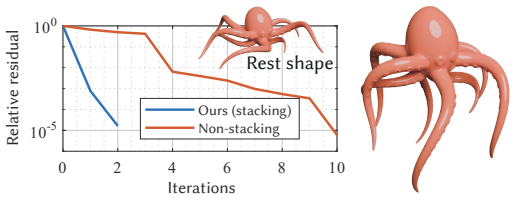

Fig. 8. Octopus simulation convergence plots show our stacking approach (blue) outperforms non-stacking subdivision (orange). By using a globally smooth residual $\varphi - G_i$, our method hits 10^{-4} residual in two iterations.

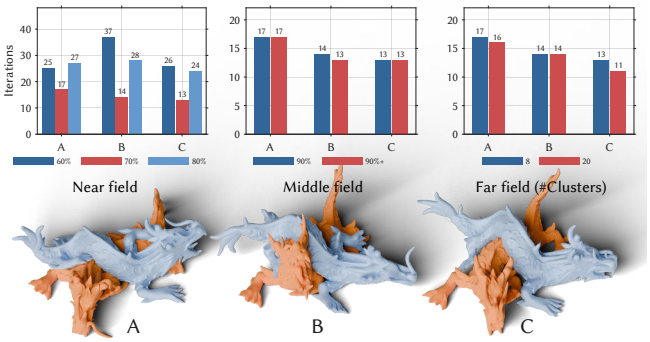

Fig. 9. (Top) Iteration counts across three representative dragon scenarios (A, B, and C) under varying domain configurations. The near-field coverage (left) is the most critical parameter, with 70% coverage consistently providing the fastest convergence; performance degrades at both lower (60%) and higher (80%) thresholds. In contrast, the middle-field expansion (center) and far-field cluster count (right) show minimal impact on iterations, indicating high robustness to parameter selection in these domains. (Bottom) Visualizations of the dragon models in various contact configurations corresponding to the test cases. "Middle = 90%+" denotes maximal expansion.

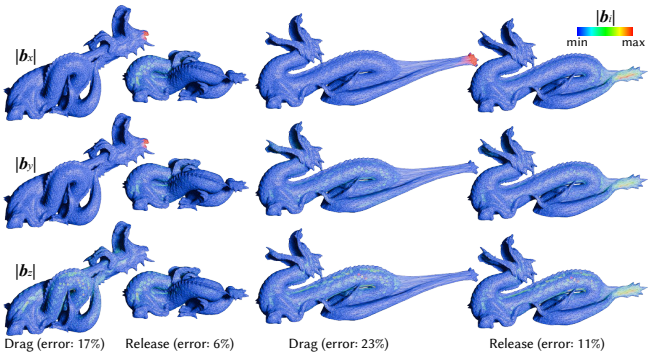

Fig. 10. Magnitude of $|\boldsymbol{b}_i|$ visualized in the x, y, and z directions. Our approach remains stable under both high-frequency (drag) and low-frequency (release) conditions, as shown by the consistent mesh integrity and magnitude distribution. Quantitative evaluation of the integration error, $\int_\Omega \varphi(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$, reveals higher relative errors (up to 23%) during high-frequency "Drag" phases, while "Release" phases exhibit lower relative errors (approximately 6%-11%) as the system returns to a lower-frequency state.

Table 1. This table summarizes the experimental setup and timing statistics for all examples discussed in the paper. **#DOF** is the total number of simulation DOFs. For multi-object scenarios, the reported statistics are **per-object** averages. Δt is the time step size. $\boldsymbol{w_j}$, $\boldsymbol{w_I}$ denotes the stiffness for material and collision constraints. We use the nodal position change $\|\Delta \boldsymbol{x}\|$ between two iterations as the convergence criterion. **#Iter.** gives the average total number of LG iterations per-time-step. **Cull|Nrw.** reports the timing for collision handling: **Cull** is for broad-phase culling; and **Nrw.** is for the narrow phase. **LG** reports the total LG iteration times. **Total** is the overall simulation times for each time step. All the timing measures are in milliseconds.

Scene	#DOF	Δt	$\boldsymbol{w_j}$, $\boldsymbol{w_I}$	$\|\Delta \boldsymbol{x}\|$	#Iter.	Cull|Nrw.	LG	Total
Tree (Fig. 1)	793K	$\frac{1}{100}$	7e7, 1e9	2e-3	19	3.5|13.7	74	95
Letters (stiffer) (Fig. 14)	534K	$\frac{1}{120}$	2e7, 1e8	2e-3	18	2.1|3.3	14	22
Letters (soft) (Fig. 14)	534K	$\frac{1}{120}$	1e6, 1e7	2e-3	12	2.1|3.9	11	19
Blender (Fig. 15)	499K	$\frac{1}{120}$	5e6, 5e7	2e-3	18	5.7|18.3	46	74
Ship (Fig. 16)	422K	$\frac{1}{100}$	1e8, 1e9	1e-3	19	11.2|52.1	56	123
Ship ×20 (Fig. 17)	8.5M	$\frac{1}{100}$	1e6, 1e9	1e-3	27	174|1, 303	2, 133	3, 749
Puffer balls (Fig. 18)	3.3M	$\frac{1}{120}$	5e5, 1e9	1e-3	17	33.4|91.2	783	912
Cloth (Fig. 19)	150K	$\frac{1}{120}$	1e4, 1e7	1e-3	42	5.7|17.8	31	57
Runway (Fig. 20)	722K	$\frac{1}{60}$	1e4, 1e7	1e-3	42	7.7|27.2	133	176
Funnel (Fig. 21)	523K	$\frac{1}{120}$	5e5, 1e7	1e-3	32	9.1|34.3	60	109

a 2-point rule. While this approach utilizes more quadrature points, it fails to match the superior convergence of our stacking method. This is because partitioning the space into multiple sub-domains introduces artificial discontinuities at the interfaces and increases the complexity of the domain management. In contrast, our stacking strategy allows the GL rule to operate on a single, globally smooth residual $\varphi - G_i$, which is significantly better suited for polynomial approximation. In Fig. 8, the octopus simulation convergence plots show our stacking approach (blue) outperforms non-stacking subdivision (orange). By using a globally smooth residual $\varphi - G_i$, our method hits 10^{-4} residual in two iterations.

Sensitivity of domain selections. The sensitivity of our multilevel quadrature to domain selection is illustrated in Fig. 9. The results indicate that the coverage of total influence within the near-field ($\int_\Omega \omega_i(\xi)\mathrm{d}\xi$) is the most critical parameter. Specifically, we observe a "U-shaped" performance trend where a 70% coverage consistently yields the fewest iterations across all test cases (A, B, and C); deviations to either higher (80%) or lower (60%) coverage lead to performance degradation. In contrast, the system exhibits significantly lower sensitivity to middle-field coverage, where "90%+" denotes maximal expansion, and the far-field cluster count, both of which have a marginal impact on convergence rates.

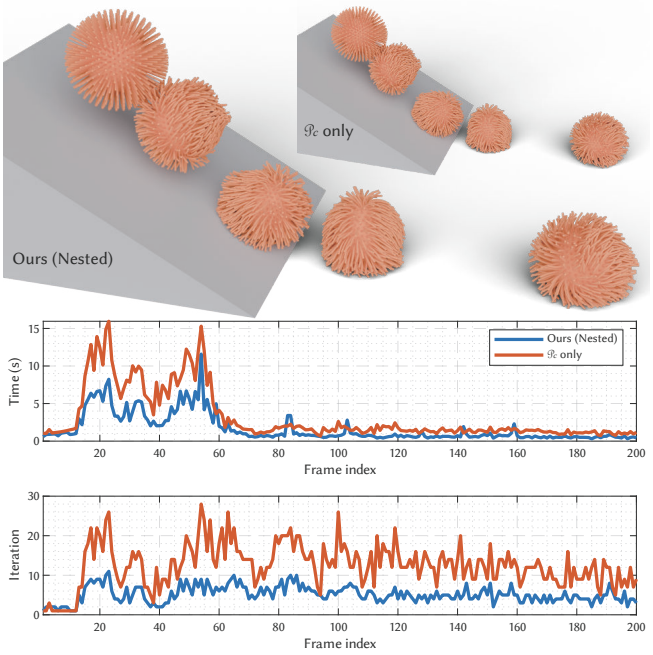

Fig. 11. Top: Sequential frames showing the rolling motion of a puffer ball. Our nested preconditioning (left) produces visually similar results compared to the "$\mathcal{P}_c$ only" (right). Bottom: Quantitative analysis of computational efficiency across 200 frames. Our method (blue) consistently outperforms the baseline (orange) in both wall-clock time per frame (top plot) and the number of iterations required for convergence (bottom plot).

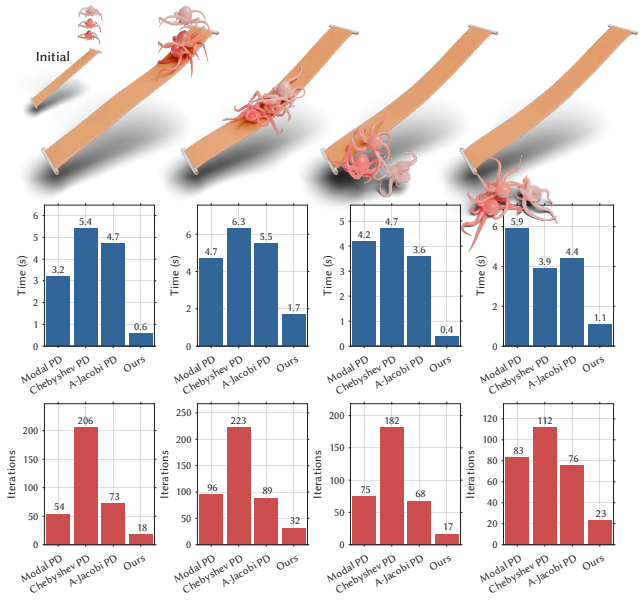

Fig. 12. We compare our method against state-of-the-art PD-based solvers by simulating three octopi dropping onto a deformable slider (top row), where all components are modeled using PD. By benchmarking across various stages of the simulation sequence, our method consistently achieves significant speedups in execution time (mid row) and requires fewer iterations to converge (bottom row) compared to existing PD methods handling complex contacts.

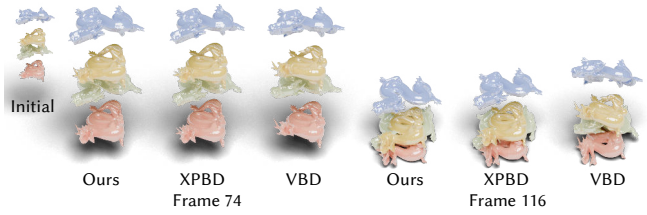

Fig. 13. We show the comparison of our method with XPBD and VBD. Our method uses a time step of $\Delta t = 1/30$ s, whereas XPBD and VBD require 50 substeps ($\Delta t = 1/1,500$ s) to remain stable. VBD suffers from significant numerical damping artifacts, while our approach efficiently preserves dynamic details at a coarser temporal discretization.

Sensitivity of $\boldsymbol{b}$'s frequency. Based on our formulation, the multi-level quadrature scheme exhibits high robustness with respect to variations in the vector field $\boldsymbol{b}$, as visualized in Fig. 10. By using the spatial prior that the impulse response $\boldsymbol{u}(i)$ behaves as a localized kernel-shaped field, we can effectively decouple the high-frequency geometric features of the integrand $\varphi(\boldsymbol{x}) = u_i(\boldsymbol{x})b(\boldsymbol{x})$ (stemming from the sharp peak of $\boldsymbol{u}(i)$) from the smoother global response (stemming from $\boldsymbol{b}(\boldsymbol{x})$). The key advantage of this approach is that as long as $\boldsymbol{b}(\boldsymbol{x})$ maintains reasonable spatial smoothness, the resulting residual field $\varphi(\boldsymbol{x}) - G_i(\boldsymbol{x})$ remains exceptionally smooth and well-suited for accurate, low-degree GL quadrature. This decoupling allows our method to remain numerically stable and accurate across both the high-frequency *drag* phase and the low-frequency *release* phase, as evidenced by the consistent mesh integrity and uniform magnitude distribution of $|\boldsymbol{b}_i|$ shown in Fig. 10. Ultimately, these results confirm that our method is largely agnostic of the specific profile of $\boldsymbol{b}(\boldsymbol{x})$, providing a principled balance between computational efficiency and physical accuracy without numerical artifacts often found in sparse sampling methods. We also evaluate the integration error, $\int_\Omega \varphi(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$, relative to the ground truth. The results indicate that the relative error increases to approximately 20% in high-frequency scenarios, whereas it remains lower (around 10%) when $\boldsymbol{b}$ represents a lower-frequency signal.

Nested preconditioning vs $\mathcal{P}_c$ preconditioning. Our nested preconditioned CG solver is tailored for scenarios with dense contacts, such as the puffer ball simulation shown in Fig. 11. By sequentially applying contact preconditioning ($\mathcal{P}_c$) to eliminate high-frequency errors and elastic preconditioning ($\mathcal{P}_e$) for structural stability, the method maintains high efficiency across the entire sequence. Compared to using $\mathcal{P}_c$ alone, the nested solver achieves a 2× speedup, reducing average per-frame computation time from 3.2 s to 1.7 s and more than doubling the convergence rate (5.2 vs. 12.5 iterations).

7.3 Comparison with other PD-based algorithms

We also compare our integration-based GPU PD solver with existing PD-based simulators, including Chebyshev PD [Wang 2015], Aggregated Jacobi (A-Jacobi) [Lan et al. 2022], and modal-subspace

Fig. 14. Our three-level quadrature scheme robustly captures both high-frequency and low-frequency propagation of φ at each node. Therefore, our method is not sensitive to the stiffness of the model. In this example, we drop letters of "Awesome simulation" into a glass bowl. Letters in the top row are 20× stiffer than those in the bottom row. There are 534K DOFs in this test, and our method robustly simulates both scenes using 22 ms (top), and 19 ms (bottom).

Fig. 15. This is another collision-heavy scenario of 36 cartoon animals inside a blender. These models are 10× stiffer than those in Fig. 21. Driven by the rotating blades, the characters undergo intense collisions and rapid agitation. Our method handles this complex scene with ease, maintaining a simulation speed of 74 ms per time step.

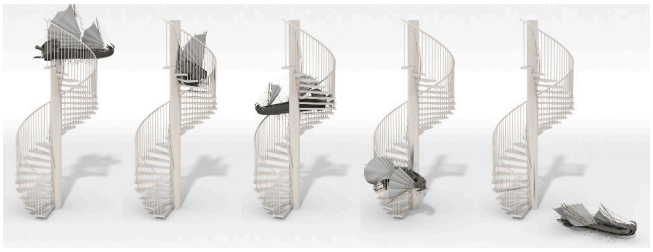

Fig. 16. In this simulation, a ship model slides alongside a spiral staircase, resulting in highly complex interactions. Our method enables high-performance simulation through the use of a nested preconditioned CG solver, processing approximately 123 ms per frame for a setup containing 422K DOFs.

preconditioned PD (Modal-PD) [Lan et al. 2024]. The test is performed by simulating three octopi dropping onto a flexible slider. We use direct-solver PD as our reference, and compare the total number of LG iterations as well as the computational time needed to solve one step along the simulation. The results are plotted in Fig. 12, and our method outperforms peer methods by a significant margin. The example shown in Fig. 12 uses a relatively large time step ($\Delta t = 1/30$ s) with a friction coefficient of $\mu = 0.3$. For a fair comparison, we launch all methods with an identical state. As such a large time step poses stability challenges for other methods across the entire simulation, we instead employ $\Delta t = 1/100$ s and $\mu = 0.1$ for the other PD variants, and compare their behaviors. Please refer to the supplemental video for details.

7.4 Comparison with XPBD and VBD

We further compare our method against XPBD and VBD, both of which are lightweight and GPU-parallelized via colored Gauss-Seidel schemes. We use a standard time step size of $\Delta t = 1/30$ s, whereas XPBD and VBD require 50 substeps ($\Delta t = 1/1,500$ s) to maintain stability for the same simulation setup (Fig. 13). Under this configuration, our method requires 37 ms per time step, achieving a 7.6× and 5.5× speedup compared to XPBD and VBD, respectively. This substantial performance gain stems from our ability to handle stiff dynamics efficiently at coarser temporal resolutions without incurring numerical explosion. In contrast, while alternative GPU-parallel solvers bypass the computational overhead of global system inversions, their reliance on local constraint projections or simplified impulses mandates severe substepping to prevent constraint violation and simulation blow-up. Furthermore, even at a fine substep size of $\Delta t = 1/1,500$ s, VBD exhibits pronounced numerical damping artifacts (see Fig. 13-VBD). This highlights a fundamental trade-off in existing lightweight formulations: forcing stability via excessive substepping or heavy damping inevitably compromises physical fidelity, whereas our method preserves high-frequency details while maintaining unconditional robustness at large time steps.

7.5 More results

Finally, we report several more complex simulation examples using our method. Fig. 14 shows snapshots of the first experiment, where we drop the letters of *Awesome simulation* into a bowl. We perform two sets of simulations: the letters on the top row are 20× stiffer than those on the bottom row. Unlike other PD-based algorithms, our method is not sensitive to the stiffness of the object. This is because our multi-level integration captures both high-frequency and low-frequency kernel influences across the model. In this experiment, there are 534K DOFs in the scene, and our method takes 19 ∼ 22 ms to simulate one time step.

Fig. 15 reports another collision-heavy example, where we put 36 cartoon animals into a blender. As the blender blades rotate, the characters are agitated and collide frequently. Our method handles this scene with ease, and the simulation runs at 74 ms per time step.

Fig. 16 showcases a high-fidelity physical simulation involving a ship model interacting with a complex static geometry (a spiral staircase). The simulation captures the ship sliding alongside

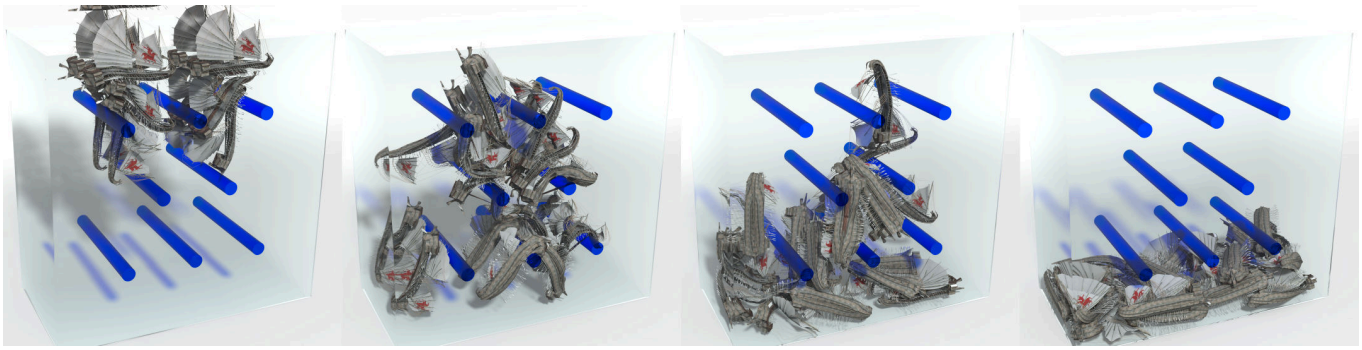

Fig. 17. When increasing the scene complexity to 20 ship models, the simulation scales effectively. Despite the scenario involving nearly 8.5M DOFs, the computational cost remains low at only 3.7 s per frame.

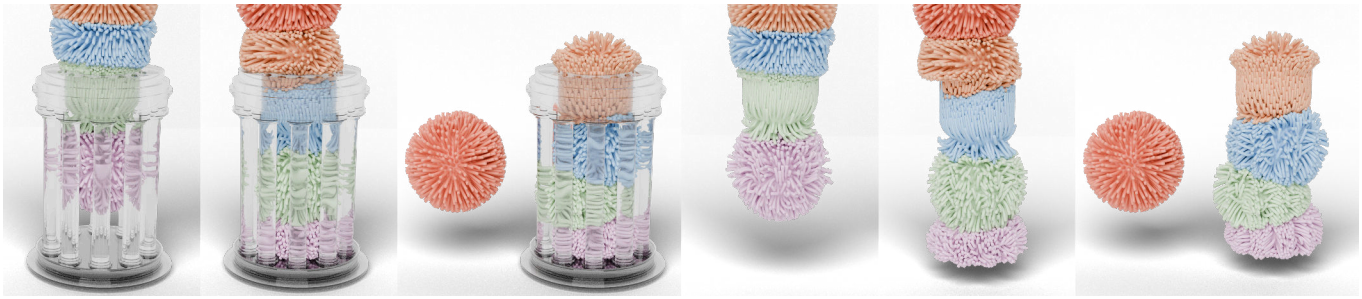

Fig. 18. In this simulation, puffer balls totaling 3.3M DOFs stack within a glass container, resulting in intricate multi-body collisions. Our method demonstrates high efficiency in this collision-intensive scenario, requiring only 912 ms per frame.

Fig. 19. For 2D manifolds, the integration domain degenerates, effectively reducing the required quadrature points compared to volumetric models. Here, five high-resolution tablecloths (totaling 150K DOFs) interact with a teapot. Our method captures intricate folding patterns in real-time, requiring only 57 ms per time step.

Fig. 20. We further evaluate our method using a cloth simulation of a runway walk. As the avatar moves, it generates complex self-collisions and cloth-avatar interactions. This setup involves 722K DOFs and achieves a performance of 176 ms per frame.

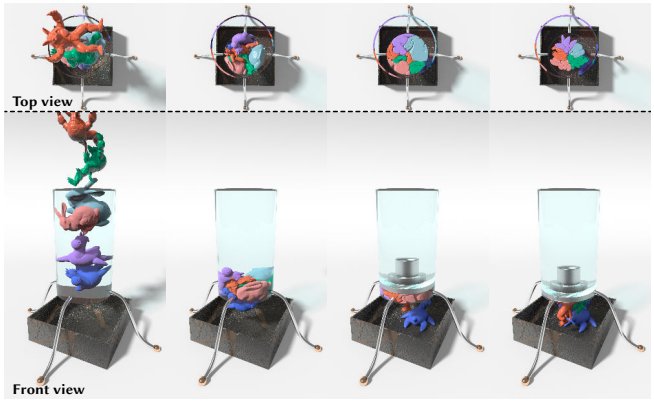

Fig. 21. Our method also performs well in scenarios involving intensive collisions and contacts. In this example, six soft bodies are placed in a funnel with a narrow bottom outlet. A heavy metal disk moves downward, pushing all objects through the funnel. There are 523K DOFs in this scene, and our method robustly completes the simulation, taking 109 ms per time step.

and through the railings and steps of a spiral staircase. This creates a high volume of highly complex interactions as the various components of the ship (hulls, masts, and sails) collide with the thin, repeating structures of the stairs. The setup is composed of 422K DOFs, indicating a high-resolution mesh capable of capturing detailed deformations or multi-body dynamics. Despite the high DOF count and interaction complexity, the method achieves a processing speed of approximately 123 ms per frame. The experiment

demonstrates the effectiveness of a nested preconditioned CG. This approach is highlighted for enabling high-performance simulation.

We evaluated the scalability and robustness of our method by significantly increasing the scene complexity, as illustrated in Fig. 17. In this experiment, the simulation was expanded to include 20 ship models interacting within a confined environment. Our simulation framework scales effectively as the number of interacting bodies and geometric complexity increase. The scenario involves nearly 8.5M DOFs, demonstrating the solver's ability to handle massive systems. Despite the extreme complexity of the scene, the computational cost remains remarkably low, requiring only 3.7 s per frame. Our method maintains physical stability and performance even when managing the dense, multi-body collisions inherent in such a high-DOF environment.

We simulate the stacking of multiple puffer balls within a glass container as shown in Fig. 18. This experiment involves approximately 3.3M DOFs and generates a massive volume of intricate multi-body collisions as the flexible bristles of the puffer balls interlock and compress. Despite the significant geometric complexity and the intensive nature of the contact constraints, our method demonstrates exceptional efficiency, processing the simulation at a rate of 912 ms per frame. The visual results in Fig. 18 confirm that our approach maintains physical stability throughout the stacking process, effectively capturing the complex deformation and collective behavior of the objects without compromising computational performance.

Our method is even more effective for cloth simulation, as the integration domain degenerates to 2D and becomes more regular. This leads to a smaller number of quadrature points. Fig. 19 shows an experiment where five pieces of high-resolution tablecloth fall onto a teapot. There are 150K DOFs in total. Our method produces high-quality cloth animation interactively, taking 57 ms per time step. We further evaluate our method using a cloth simulation of a runway walk, as illustrated in Fig. 20. As the avatar moves, the simulation successfully generates complex self-collisions and intricate cloth-avatar interactions. This specific setup involves 722K DOFs and achieves a performance of 176 ms per frame, demonstrating the efficiency of our approach in dynamic scenarios.

Fig. 21 shows another challenging simulation for PD-based solvers. In this test, we put six soft 3D objects into a funnel-like container. After all objects reach the bottom, we push them out of the funnel with a metal disk. There are 523K DOFs in the scene, involving extensive static contacts among the models and the collider. Our method efficiently simulates the scene, taking 109 ms for each time step.

A more complex simulation is reported in the teaser figure, where the tree branches consist of about 800K DOFs. To further enrich the visual effects, we also attach 23.5K deformable leaves (which do not participate in collision detection). Our method simulates this scene at an interactive rate, requiring only 95 ms per step.

8 Conclusion

We have presented a novel integration-based perspective to resolve the computational bottleneck in PD. By interpreting dense vector products as integrals, we expose the spatial structure of the underlying impulse-response kernels. This view leads to a multi-level quadrature strategy that approximates near, middle, and far fields differently, yielding a sparse yet accurate estimate of the global solve, and preserving GPU-friendly parallelism. In collision-free settings, a smaller number of iterations provides convergence comparable to a direct PD solve; while in contact-rich scenarios, our nested preconditioner combines the quadrature-based collision-free solve with a lightweight contact solve to accelerate PCG for the time-varying global system. Across high-resolution deformable simulations, our experiments show that this integration-oriented sparsification enables robust convergence and substantial speedups over existing PD-based solvers.

Our method is most effective when the collision-free system matrix remains fixed. If material parameters, boundary conditions, or time step sizes vary, the quadrature distribution must be recomputed. Consequently, our method is not well-suited for scenarios requiring real-time user adjustment of material parameters or dynamic updates to boundary conditions.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. Yin Yang and Yuqi Meng are partially supported by NSF 2301040. Weiwei Xu and Lei Lan are partially supported by NSFC (92570206, 62421003) and the State Key Laboratory of CAD&CG, Zhejiang University. We also acknowledge support from Envora.

References

Steven S An, Theodore Kim, and Doug L James. 2008. Optimizing cubature for efficient integration of subspace deformations. *ACM transactions on graphics (TOG)* 27, 5 (2008), 1–10.

David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (New Orleans, Louisiana) *(SODA '07)*. Society for Industrial and Applied Mathematics, USA, 1027–1035.

David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98 (SIGGRAPH '98)*. ACM Press, 43–54. doi:10.1145/280814.280821

Jernej Barbič and Doug L. James. 2005. Real-Time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Transactions on Graphics* 24, 3 (July 2005), 982–990. doi:10.1145/1073204.1073300

Josh Barnes and Piet Hut. 1986. A hierarchical O(N log N) force-calculation algorithm. *Nature* 324, 6096 (1986), 446–449. doi:10.1038/324446a0

Klaus-Jürgen Bathe. 2006. *Finite element procedures*. Klaus-Jurgen Bathe.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics: fusing constraint projections for fast simulation. *ACM Transactions on Graphics* 33, 4 (July 2014), 1–11. doi:10.1145/2601097.2601116

Christopher Brandt, Elmar Eisemann, and Klaus Hildebrandt. 2018. Hyper-reduced projective dynamics. *ACM Transactions on Graphics* 37, 4 (July 2018), 1–13. doi:10.1145/3197517.3201387

Susanne C. Brenner and L. Ridgway Scott. 2008. *The Mathematical Theory of Finite Element Methods*. Springer New York. doi:10.1007/978-0-387-75934-0

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

He Chen, Elie Diaz, and Cem Yuksel. 2023. Shortest path to boundary for self-intersecting meshes. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–15.

Min Gyu Choi and Hyeong-Seok Ko. 2005. Modal Warping: Real-Time Simulation of Large Rotational Deformation and Manipulation. *IEEE Transactions on Visualization and Computer Graphics* 11, 01 (Jan. 2005), 91–101. doi:10.1109/tvcg.2005.13

Mathieu Desbrun and Marie-Paule Gascuel. 1996. *Smoothed Particles: A new paradigm for animating highly deformable bodies*. Springer Vienna, 61–76. doi:10.1007/978-3-7091-7486-9_5

Lawrence C. Evans. 2010. *Partial Differential Equations* (2nd ed.). Graduate Studies in Mathematics, Vol. 19. American Mathematical Society, Providence, RI.

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Transactions on Graphics* 35, 6 (Nov. 2016), 1–9. doi:10.1145/2980179.2982437

Chris Giles, Elie Diaz, and Cem Yuksel. 2025. Augmented Vertex Block Descent. *ACM Transactions on Graphics* 44, 4 (July 2025), 1–12. doi:10.1145/3731195

Teofilo F Gonzalez. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical computer science* 38 (1985), 293–306.

Dewen Guo, Minchen Li, Yin Yang, Sheng Li, and Guoping Wang. 2024. Barrier-Augmented Lagrangian for GPU-based Elastodynamic Contact. *ACM Transactions on Graphics* 43, 6 (Nov. 2024), 1–17. doi:10.1145/3687988

Dewen Guo, Zixuan Lu, Zhiyong He, Yuqi Meng, Bohan Wang, Lei Lan, Weiwei Xu, Chenfanfu Jiang, and Yin Yang. 2026a. JGS2-GQ: Training-free 2nd Jacobi with Gaussian Quadrature. *ACM Trans. Graph.* 45, 4, Article 123 (July 2026), 18 pages. doi:10.1145/3811274

Dewen Guo, Ran Tian, Sinuo Liu, Guoping Wang, and Sheng Li. 2025a. Diagonal Hessian Proxy for Efficient Elastic Simulation Using Peridynamics. *IEEE Transactions on Visualization and Computer Graphics* 31, 9 (Sep. 2025), 6466–6483. doi:10.1109/TVCG.2024.3516480

Dewen Guo, Zhendong Wang, Minchen Li, Sheng Li, Guoping Wang, Huamin Wang, Chenfanfu Jiang, and Yin Yang. 2026b. Heterogeneous Subspace Corrections for GPU Deformable Multibody Dynamics. *ACM Trans. Graph.* 45, 4, Article 100 (July 2026), 16 pages. doi:10.1145/3811275

Dewen Guo, Zhendong Wang, Zegao Liu, Sheng Li, Guoping Wang, Yin Yang, and Huamin Wang. 2025b. Fast Physics-Based Modeling of Knots and Ties using Templates. In *ACM SIGGRAPH 2025 Conference Papers (SIGGRAPH Conference Papers '25)*. Association for Computing Machinery, New York, NY, USA, Article 43, 9 pages. doi:10.1145/3721238.3730622

Dewen Guo, Zhendong Wang, Zegao Liu, Sheng Li, Guoping Wang, Yin Yang, and Huamin Wang. 2025c. Progressive Outfit Assembly and Instantaneous Pose Transfer. In *SIGGRAPH Asia 2025 Conference Papers* (Hong Kong, China) *(SA Conference papers '25)*. Association for Computing Machinery, New York, NY, USA, 12 pages. doi:10.1145/3757377.3763868

Tieding Guo and Giuseppe Rega. 2023. Reduced-order modeling of geometrically nonlinear structures. Part II: Correspondence and unified perspectives on different reduction techniques. *Nonlinear Dynamics* 111, 21 (Oct. 2023), 19655–19684. doi:10.1007/s11071-023-08745-8

Houde Han and Dongsheng Yin. 2026. *Boundary Value Problems of Modified Helmholtz Equation*. Springer Nature Singapore, 97–127. doi:10.1007/978-981-95-1088-7_4

Zhiyong He, Dewen Guo, Minghao Guo, Yili Zhao, Wojciech Matusik, Hao Su, Chenfanfu Jiang, Peter Yichen Chen, and Yin Yang. 2026. M-ABD: Scalable, Efficient, and Robust Multi-Affine-Body Dynamics. *ACM Transactions on Graphics* 45, 4 (July 2026). doi:10.1145/3811276

Kemeng Huang, Floyd M. Chitalu, Huancheng Lin, and Taku Komura. 2024. GIPC: Fast and Stable Gauss-Newton Optimization of IPC Barrier Energy. *ACM Transactions on Graphics* 43, 2 (March 2024), 1–18. doi:10.1145/3643028

Hughes. 2000. *The Finite Element Method*. Dover Publications, Mineola, NY.

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic coordinates for character articulation. *ACM Transactions on Graphics* 26, 3 (July 2007), 71. doi:10.1145/1276377.1276466

Ryo Kikuuwe, Hiroaki Tabuchi, and Motoji Yamamoto. 2009. An edge-based computationally efficient formulation of Saint Venant-Kirchhoff tetrahedral finite elements. *ACM Transactions on Graphics* 28, 1 (Jan. 2009), 1–13. doi:10.1145/1477926.1477934

Lei Lan, Zixuan Lu, Jingyi Long, Chun Yuan, Xuan Li, Xiaowei He, Huamin Wang, Chenfanfu Jiang, and Yin Yang. 2024. Efficient GPU cloth simulation with non-distance barriers and subspace reuse. *arXiv preprint arXiv:2403.19272* (2024).

Lei Lan, Zixuan Lu, Chun Yuan, Weiwei Xu, Hao Su, Huamin Wang, Chenfanfu Jiang, and Yin Yang. 2025. JGS2: Near Second-order Converging Jacobi/Gauss-Seidel for GPU Elastodynamics. *ACM Transactions on Graphics* 44, 4 (July 2025), 1–15. doi:10.1145/3731183

Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022. Penetration-free projective dynamics on the GPU. *ACM Transactions on Graphics* 41, 4 (July 2022), 1–16. doi:10.1145/3528223.3530069

Chunlei Li, Peng Yu, Tiantian Liu, Siyuan Yu, Yuting Xiao, Shuai Li, Aimin Hao, Yang Gao, and Qinping Zhao. 2025. MGPBD: A Multigrid Accelerated Global XPBD Solver. In *ACM SIGGRAPH 2025 Conference Papers (SIGGRAPH Conference Papers '25)*. ACM, 1–11. doi:10.1145/3721238.3730720

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

Xuan Li, Yu Fang, Lei Lan, Huamin Wang, Yin Yang, Minchen Li, and Chenfanfu Jiang. 2023. Subspace-Preconditioned GPU Projective Dynamics with Contact for Cloth Simulation. In *SIGGRAPH Asia 2023 Conference Papers (SA '23)*. ACM, 1–12. doi:10.1145/3610548.3618157

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-newton methods for real-time simulation of hyperelastic materials. *Acm Transactions on Graphics (TOG)* 36, 3 (2017), 1–16.

Zixuan Lu, Ziheng Liu, Lei Lan, Huamin Wang, Yuko Ishiwaka, Chenfanfu Jiang, Kui Wu, and Yin Yang. 2025. High-performance CPU Cloth Simulation Using Domain-decomposed Projective Dynamics. *ACM Transactions on Graphics* 44, 4 (July 2025), 1–17. doi:10.1145/3731182

Mickaël Ly, Jean Jouve, Laurence Boissieux, and Florence Bertails-Descoubes. 2020. Projective dynamics with dry frictional contact. *ACM Transactions on Graphics* 39, 4 (Aug. 2020). doi:10.1145/3386569.3392396

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games (MiG '16)*. ACM, 49–54. doi:10.1145/2994258.2994272

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-based elastic materials. In *ACM SIGGRAPH 2011 papers (SIGGRAPH '11)*. ACM, 1–8. doi:10.1145/1964921.1964967

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

Matthew Overby, George E. Brown, Jie Li, and Rahul Narain. 2017. ADMM Projective Dynamics: Fast Simulation of Hyperelastic Models with Dynamic Constraints. *IEEE Transactions on Visualization and Computer Graphics* 23, 10 (Oct. 2017), 2222–2234. doi:10.1109/TVCG.2017.2730875

James F. O'Brien and Jessica K. Hodgins. 1999. Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99 (SIGGRAPH '99)*. ACM Press, 137–146. doi:10.1145/311535.311550

A. Pentland and J. Williams. 1989. Good vibrations: modal dynamics for graphics and animation. (July 1989), 215–222. doi:10.1145/74333.74355

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 2007. *Numerical Recipes 3rd Edition: The Art of Scientific Computing* (3 ed.). Cambridge University Press, USA.

Xavier Provot. 1995. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. In *Proceedings of Graphics Interface '95*. 147–154.

Jie Shen and Yee-Hong Yang. 1998. Deformable Object Modeling Using the Time-Dependent Finite Element Method. *Graphical Models and Image Processing* 60, 6 (Nov. 1998), 461–487. doi:10.1006/gmip.1998.0484

Eftychios Sifakis and Jernej Barbič. 2012. FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses (SIGGRAPH '12)*. ACM, 1–50. doi:10.1145/2343483.2343501

Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. 2015. Subspace condensation: full space adaptivity for subspace deformations. *ACM Transactions on Graphics* 34, 4 (July 2015), 1–9. doi:10.1145/2766904

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically Deformable Models. *ACM SIGGRAPH Computer Graphics* 21, 4 (1987), 205–214. doi:10.1145/37402.37427

Christoph von Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. 2013. An efficient construction of reduced deformable objects. *ACM Transactions on Graphics* 32, 6 (Nov. 2013), 1–10. doi:10.1145/2508363.2508392

Huamin Wang. 2015. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics* 34, 6 (Nov. 2015), 1–9. doi:10.1145/2816795.2818063

Huamin Wang and Yin Yang. 2016. Descent methods for elastic body simulation on the GPU. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–10.

Keith Waters. 1987. A muscle model for animation three-dimensional facial expression. *ACM SIGGRAPH Computer Graphics* 21, 4 (Aug. 1987), 17–24. doi:10.1145/37402.37405

Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. 2015. Expediting precomputation for reduced deformable simulation. *ACM Transactions on Graphics* 34, 6 (Nov. 2015), 1–13. doi:10.1145/2816795.2818089

Chang Yu, Xuan Li, Lei Lan, Yin Yang, and Chenfanfu Jiang. 2024. XPBI: Position-Based Dynamics with Smoothing Kernels Handles Continuum Inelasticity. In *SIGGRAPH Asia 2024 Conference Papers (SA '24)*. ACM, 1–12. doi:10.1145/3680528.3687577

Chun Yuan, Zixuan Lu, Haoyang Shi, Dewen Guo, Huamin Wang, Chenfanfu Jiang, Zherong Pan, Kui Wu, and Yin Yang. 2026. Interactive Yarn-level Knitwear with Nested Douglas-Rachford Splitting. *ACM Trans. Graph.* 45, 4, Article 103 (July 2026), 22 pages. doi:10.1145/3811277