JGS2-GQ: Training-free 2nd Jacobi with Gaussian Quadrature

DEWEN GUO, University of Utah, USA
ZIXUAN LU, University of Utah, USA
ZHIYONG HE, University of Utah, USA
YUQI MENG, University of Utah, USA
BOHAN WANG, National University of Singapore, Singapore
LEI LAN, Zhejiang University, China
WEIWEI XU, Zhejiang University, China
CHENFANFU JIANG, University of California Los Angeles, USA
YIN YANG, University of Utah, USA

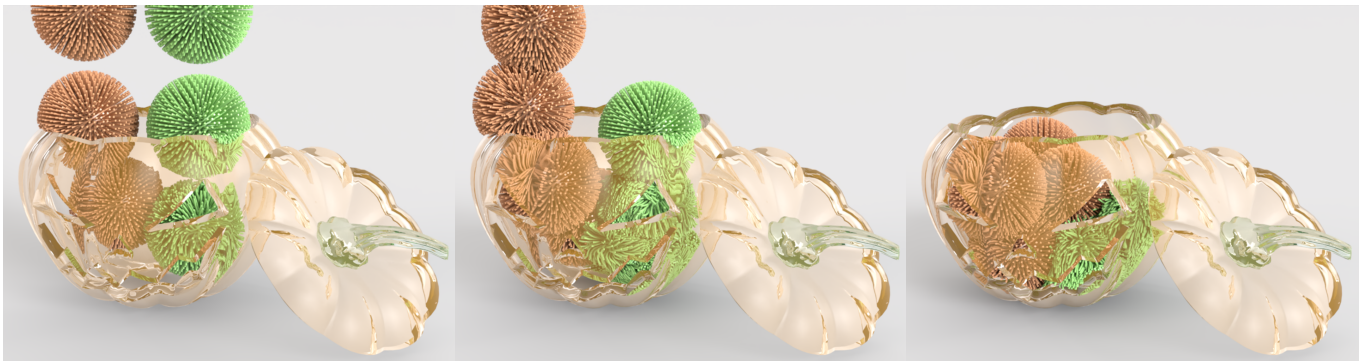
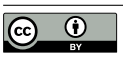


Fig. 1. **Puffer balls in the pumpkin.** We propose JGS2-GQ, a GPU-based high-performance algorithm for simulating high-resolution deformable models. JGS2-GQ inherits the principles of JGS2 by incorporating global information into local Newton subproblems at mesh nodes. The core innovation is a new modality for evaluating reduced gradients and Hessians: instead of data-driven training, JGS2-GQ exploits the local polynomiality of low-dimensional subspace integration and employs a Gaussian quadrature scheme. This approach obviates data preparation and training by approximating the integrand rather than the integral. Consequently, JGS2-GQ achieves superior robustness, particularly for novel and high-frequency deformations. It maintains the massive parallelizability of the Jacobi method while converging nearly as fast as Newton's method. The teaser figure showcases these advantages: ten high-resolution puffer balls falling into a pumpkin, where dense hairs undergo highly curved local deformations upon contact. The orange balls are 20× stiffer than the green ones. We use IPC barriers [Li et al. 2020] coupled with CCD-based line search filtering to ensure that all the primitives are free of interpenetration. Such stiffness disparity between colliding models with IPC is particularly challenging for GPU-based solvers. JGS2-GQ robustly handles this complex scene of 5.4M DOFs at 742 ms per time step ($\Delta t = 0.01\ sec$). It is 74.9× faster than the existing GPU-based IPC method e.g., [Guo et al. 2024]. Compared with vanilla JGS2 [Lan et al. 2025], our method converges 40% faster if the Cubature training is carried out with low-frequency eigen modes.

JGS2 is a Jacobi-like GPU simulation algorithm. It avoids the overshooting issue by augmenting each subproblem with a perturbation subspace that predicts the global influence of the local solve. The efficiency of JGS2 is due to Cubature-based subspace integration at each subproblem. Being a data-driven method, Cubature requires a set of representative deformed poses that cover deformations likely to occur in the simulation. This requirement is unlikely for high-resolution deformation with rich local details. Therefore, simulation performance and convergence degenerate when Cubature extrapolates. This paper proposes a training-free subspace integration algorithm based on classic Gaussian quadrature (GQ). We leverage the fact that the subproblem's subspace bases can be well-approximated by a low-degree multivariable polynomial, which suggests GQ an excellent candidate for Cubature substitute. To this end, we introduce a novel algorithm that adaptively generates the integration region for each subproblem. As a result, GQ integration can be analytically retrieved without cumbersome data generation and training. We also show how to handle frictional contact by modifying the pre-computed perturbation subspace. The resulting JGS2-GQ framework is more versatile than the vanilla JGS2 method. It is more stable for large and novel deformations, and is free of data generation and expensive training, while maintaining a near second-order convergence that is comparable to Newton. Performance-wise, JGS2-GQ is as efficient as JGS2, which is three

Authors' Contact Information: Dewen Guo, University of Utah, Salt Lake City, USA, guodw.sh@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; Yuqi Meng, University of Utah, Salt Lake City, USA, aressegetesstery@gmail.com; Bohan Wang, National University of Singapore, Singapore, bh.wang@nus.edu.sg; Lei Lan, Zhejiang University, Hangzhou, China, lanlei.virhum@gmail.com; Weiwei Xu, Zhejiang University, Hangzhou, China, xww@cad.zju.edu.cn; Chenfanfu Jiang, University of California Los Angeles, 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 1557-7368/2026/7-ART123
https://doi.org/10.1145/3811274

orders faster than classic CPU methods and up to two orders faster than classic GPU algorithms. When novel deformation occurs, JGS2-GQ outperforms JGS2 over 50%.

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

Additional Key Words and Phrases: GPU algorithm, Deformable model, Numerical integration, Parallel optimaization

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

1 Introduction

Graphics processing units (GPUs) serve as a primary catalyst for modern computer graphics. With the advancement of GPGPU techniques, high-resolution physical simulations increasingly turn to high throughput from hardware parallelization. Departing from traditional CPU solvers, GPU-based schemes often favor parallel iterative methods, trading convergence for a (much) lighter per-iteration cost. A popular category of GPU-driven simulation frameworks relies on block-coordinate-descent-like variants. The core strategy is to decompose the global system into decoupled subproblems defined over individual domains, elements, or nodes. Because each subproblem involves a limited number of degrees of freedom (DOFs), its reduced complexity enables massive parallel solves. Both linear and non-linear Gauss-Seidel (GS) and Jacobi methods are classic representatives of this approach. Over the past decade, many seminal works have followed this overarching idea and delivered impressive results.

A dilemma of GPU simulation is the incompatibility between convergence and parallelism. Subproblems are typically kept small for better utilization of GPU architecture and maximized hardware concurrency. On the other hand, such fine-grained decomposition comes at the cost of convergence, especially for high-resolution models or stiff simulations. The underlying reason is subproblem *overshoot* i.e., a locally optimal update within a subproblem almost always conflicts with the global equilibrium. Recently, Lan et al. [2025] presented an effective remedy to overshoot by constructing a low-dimensional perturbation subspace. This subspace characterizes how local displacement changes propagate to other DOFs, which prevents local updates from adversely undermining global convergence. As a result, Jacobi or GS iterations achieve convergence comparable to Newton's method, and this framework is therefore named as JGS2.

The key enabler of the combined convergence and efficiency in JGS2 is Cubature sampling [An et al. 2008]. In JGS2, each subproblem is essentially a customized reduced model rather than an isolated node or element. It is well-known that the runtime performance bottleneck of such reduced simulations lies in the subspace integration of the gradient and Hessian. Cubature represents the state-of-the-art sparse sampling technique, which has been the standard acceleration solution for reduced integration. However, as a data-driven sampling scheme, Cubature inevitably inherits the limitations of learning-based methods. It is data-dependent, necessitating a set of *representative* deformation poses. Unlike general reduced simulations, subproblems in JGS2 capture local and high-frequency deformations, making the integration accuracy more sensitive to the training samples. While Cubature yields high-quality results when the current deformation resembles the training set, it can become inaccurate or even cause the simulation to diverge when encountering unseen poses. Another issue is the training overhead. Because Cubature must simultaneously optimize sampling points and weights, each iteration involves estimating multiple sampling candidates via a non-negative least-square (NNLS) fitting. This training must be performed for every subproblem, which could scale to hundreds of thousands or even millions in complex models, making the pre-computation phase excessively heavy.

In this paper, we propose a training-free reduced integration strategy tailored for JGS2. Instead of resorting to data-driven training, we leverage the fact that the perturbation subspace at each subproblem is smooth and can be *locally* well approximated by low-degree multivariable polynomials. Hence, we can use discontinuous and piecewise polynomial fields to efficiently estimate the reduced gradient/Hessian. We follow the Gaussian quadrature (GQ) scheme by setting the sampling points at the roots of Legendre polynomials, and name this new algorithm JGS2-GQ. Based on the distribution of the subspace at each subproblem, we adaptively partition the integration domain i.e., a 3D deformable model into multiple rectangular regions, named cuboids, to facilitate 3D Gaussian integration. Intuitively, our method evaluates reduced integration via a dedicated set of hexahedral elements of different orders for each subproblem. As the positions and weights of GQ sampling points are analytically available, our sparse integration does not need the support of training deformations from nonlinear modal analysis or pre-simulation, and it offers much more robust performance than Cubature for novel deformations.

In addition to the GQ-based subspace integration method, we also present a new algorithm for JGS2-GQ to adjust the perturbation subspace under frictional contacts. In particular, we model the body-level strain propagation via collision as a two-stage coupling: the coupling through the colliding primitives (e.g., node-triangle or edge-edge pairs), and the coupling on the body via elasticity. The original collision-free perturbation subspace can then be adjusted accordingly to account for such interactions.

We have tested our method in a wide range of complex, collision-intensive, and large-scale simulations. JGS2-GQ offers valuable improvements over the vanilla JGS2 framework. It is as parallelizable as Jacobi or JGS2 and possesses a strong, near-second-order convergence rate similar to Newton's method. Meanwhile, it is free of any data generation and training, and it exhibits better convergence (up to 84×) under frictional contacts than JGS2. Compared with other state-of-the-art GPU-based simulation algorithms, JGS2-GQ remains orders of magnitude faster.

2 Related Work

2.1 Deformable body simulation

Stable simulation of high-resolution deformable bodies commonly relies on implicit Euler time integration with Newton's method applied at each timestep. Whether formulated directly in terms of

forces [Baraff and Witkin 1998] or as an optimization problem derived from elastic potentials [Martin et al. 2011], this approach leads to solving a large, globally coupled yet sparse linear system whose size is proportional to the number of DOFs at each iteration. For large-scale simulations, repeated linear solves dominate the computational cost and become the primary performance bottleneck.

The most straightforward category of improvements optimize within the framework of Newton's method, either by accelerating linear solves or by estimating effective search directions without solving the full linear system. Observing that local relaxations efficiently eliminate high-frequency error, and the low-frequency error becomes high-frequency when viewed on a coarser grid, Bolz et al. [2003] proposed a multi-grid solver achieving faster convergence for linear solves without altering the underlying physical model. Multiple elasticity-aware multigrid transfer operators have been studied later for accommodation of larger simulation scale and codimensional geometry [Tamstorf et al. 2015; Xian et al. 2019; Zhang et al. 2024; Zhu et al. 2010]. Alternatively, Hecht et al. [2012] reuses previous factorization results to reduce amortized per-iteration cost of linear solving. Other methods reduce per-iteration cost by reusing information across Newton steps, exploiting elastic energy structure or recent gradient history to form efficient descent directions and, in some cases, avoid explicit linear solves altogether [Li et al. 2019; Liu et al. 2017]. Beyond optimization of linear solvers, Chen et al. [2024a] seek improvements on convergence of projected Newton's method by spectral filtering, suppressing ill-conditioned eigenmodes. Longva et al. [2020] enables accurate, high-order finite element embedding via quadrature. This work introduces a general-purpose framework for high-order finite element embedding by combining a geometry-independent quadrature generation method with an adapted Additive-Schwarz preconditioner to ensure both integration accuracy and numerical stability.

2.2 Constraint-based formulation

Other prior works approach the linear solving bottleneck differently, by simplifying the underlying elasticity model by reformulating dynamics in terms of *constraints*, resulting in time-invariant system matrix or avoiding linear solving altogether. Position-based dynamics (PBD) [Müller et al. 2007] advances simulation through explicit position prediction followed by iterative constraint projections, avoiding force integration and global linear solves, which yields high stability and efficiency for real-time applications but results in heuristic, iteration-dependent stiffness. Extended PBD (XPBD) [Macklin et al. 2016] introduces compliance parameters that restore physical meaning and eliminate iteration dependence, with further extensions incorporating plasticity and multi-grid acceleration to improve expressiveness and convergence [Li et al. 2025; Yu et al. 2024]. Rather than explicitly enforcing constraints, Projective dynamics (PD) [Bouaziz et al. 2014] formulates simulation as an optimization problem solved by alternating local projections and global quadratic minimization, enabling highly parallel local updates and efficient global solves via constant-Hessian prefactorization or PCG. Subsequent work has expanded PD with improved parallel solvers, richer energy models, model reduction, and domain decomposition [Brandt et al. 2018; Fratarcangeli et al. 2016; Lu et al. 2025; Narain et al. 2017; Wang 2015], as well as robust collision and contact handling through barrier-based formulations and specialized preconditioning [Lan et al. 2022a; Li et al. 2023; Ly et al. 2020].

2.3 Subspace simulation

In parallel to model simplification via constraint reformulation, another line of work seeks to reduce the cost of implicit simulation by lowering the effective dimension of the linear systems to be solved. Instead of operating directly in full simulation DOFs, these methods project simulation onto a low-dimensional subspace. An algebraic way of reducing system dimension is via *model reduction*, where the low-dimensional subspace is determined through analysis of system matrix, which captures the dominant deformation nodes [Barbič and James 2005; Choi and Ko 2005; Pentland and Williams 1989]. Several improvements on the choice of subspace have been developed to avoid artificial stiffening due to reduced deformation space, such as modal derivatives [Barbič and James 2005], geometric modal derivatives [von Tycowicz et al. 2013], or dynamic subspace adaptation [Teng et al. 2015]. Projection cost can be further reduced using cubature sampling, which approximates subspace integration with a small set of weighted elements [An et al. 2008].

Alternatively, system size can be reduced explicitly by introducing a low-dimensional geometric control space. Sederberg and Parry [1986] embeds detailed geometry into a uniform coarse lattice, where free-form deformation is driven by a small number of lattice control points. To better accommodate complex geometries and enable spatially varying control resolution, Huang et al. [2008] extends this idea to coarse tetrahedral control mesh embeddings, which conform more naturally to the underlying shape. Deformation graph representations further improve geometric adaptability by expressing deformation through a sparse set of local affine transformations defined on graph nodes [Sumner et al. 2007]. Beyond the construction of control geometry, a complementary line of work focuses on how deformation is interpolated from the control space to the high-resolution mesh: within the cage-based framework, various coordinate-based interpolation schemes have been proposed, including mean value [Ju et al. 2005], harmonic [Joshi et al. 2007], and Green coordinates [Lipman et al. 2008]. While such geometric reduction alleviates artificial stiffening commonly observed in model reduction, it typically smooths out fine-scale local details.

2.4 Parallelization

With recent advances in hardware, exploiting parallelism on multicore CPUs and GPUs has become an increasingly important direction for accelerating simulation. A straightforward idea to address the linear solving bottleneck would be to split the linear system into several independent smaller systems, with constraints enforced on boundaries. Farhat and Roux [1991] introduces a dual approach using Lagrangian multipliers to enforce continuity across subdomains, with an extension incorporating a dual-primal coarse space for improved conditioning and scalability [Farhat et al. 2001]. [Mandel 1993] instead adopts a primal variant, enforcing continuity via coarse corrections with error balancing. This formulation later evolved into Dohrmann [2003] for scalability.

However, domain decomposition methods are poorly suited to GPUs, as each subdomain requires solving moderate-sized linear systems with irregular memory access and synchronization overhead. Several works have adapted Newton-based formulations to GPU architectures through eigenanalysis of Hessian structures and improving conditioning through inexact Newton-PCG solvers and GPU-friendly sparse representations [Guo et al. 2024; Huang et al. 2024]. However these methods still rely on synchronization-heavy globally coupled linear solves. This motivates GPU-native formulations that completely avoid global solves.

Motivated by these limitations, recent work seeks GPU-native formulations that eliminate global linear solves entirely. Since domain decomposition still involves solving moderately sized linear systems that are costly on GPUs, a natural progression is to further refine the partitioning until each subdomain becomes small enough to admit inexpensive GPU execution, which ultimately leads to per-vertex local updates. A representative work is Vertex Block Descent (VBD) [Chen et al. 2024b], which performs inexpensive Gauss-Seidel-style updates on one vertex position per iteration and is therefore trivially parallelizable. However, as each update ignores contributions from most other vertices, convergence can be slow for strongly coupled systems. Plainly integrating over all other vertices will not work either, as this will involve accumulating contributions of the entire geometry, causing per-update cost scaling with system size and quickly harming performance due to GPU memory bandwidth. To address this limitation, Lan et al. [2025] introduces a cubature-based sparse integration scheme combined with a corotated subspace formulation. By factoring out per-node rigid rotations, the method ensures compatibility with the original cubature approach while enabling more efficient and accurate handling of large deformations in vertex-based solvers. With appropriate selection of training poses, this approach achieves near-second-order convergence. The stability and accuracy of data-driven sparse integration schemes, and the resulting convergence rate, however, is highly dependent on the quality and representativeness of the training data. This limitation motivates alternative formulations that preserve locality and sparse integration while relying on well-defined, non-data-driven integration rules.

3 Second-order Jacobi

To make the paper self-contained, we start with a brief review of JGS2 [Lan et al. 2025]. JGS2 is a GPU-based simulation framework, and its overall computational procedure is similar to conventional Jacobi (or GS) methods, which solve sub-instances of the total variational optimization at mesh nodes in parallel.

Suppose that we concern the elastodynamics of a deformable object such as a soft body or a piece of garment with the implicit Euler. At each time step $t+1$, we solve the variational optimization [Kane et al. 2000] of:

$$\boldsymbol{u}^{t+1} = \arg\min_{\boldsymbol{u}} E,\ E = \frac{1}{2\Delta t^2}\|\boldsymbol{u} - (\hat{\boldsymbol{u}} + \Delta t\hat{\dot{\boldsymbol{u}}} + \Delta t^2 \boldsymbol{M}^{-1}\boldsymbol{f}_{ext})\|_{\boldsymbol{M}}^2 + \Psi(\boldsymbol{u}), \tag{1}$$

where $\boldsymbol{u} \in \mathbb{R}^{3N}$ is the unknown vector of the mesh displacement with N being the total number of nodes on the model. $\hat{\boldsymbol{u}}$ and $\hat{\dot{\boldsymbol{u}}}$ are the known displacement and velocity vectors from the previous time step. $\boldsymbol{f}_{ext}$ is the external force; $\boldsymbol{M}$ is the constant mass matrix; and Δt is the time step size.

To find $\boldsymbol{u}^{t+1}$ in Eq. (1), each (global) Newton iteration solves a linearized system for a displacement increment $\delta\boldsymbol{u}$:

$$\boldsymbol{H}^{(k)}\delta\boldsymbol{u}^{(k)} = -\boldsymbol{g}^{(k)}, \tag{2}$$

where $\boldsymbol{g}^{(k)} = \frac{\partial E}{\partial \boldsymbol{u}}\big|_{\boldsymbol{u}=\boldsymbol{u}^{(k)}}$ and $\boldsymbol{H}^{(k)} = \frac{\partial^2 E}{\partial \boldsymbol{u}^2}\big|_{\boldsymbol{u}=\boldsymbol{u}^{(k)}}$ are the global gradient and Hessian of E. The update of $\boldsymbol{u}^{(k+1)} \leftarrow \boldsymbol{u}^{(k)} + \delta\boldsymbol{u}^{(k)}$ is expected to lower E quadratically if $\boldsymbol{u}^{(k)}$ is not far away from $\boldsymbol{u}^*$ i.e., the actual minimizer of E.

When a Jacobi-like GPU parallelization is invoked, Eq. (1) is decomposed into so-called subproblems at each node i: $\arg\min_{\boldsymbol{u}_i} E(\boldsymbol{u}_i)$. Each subproblem is of low dimension and can therefore be solved efficiently using local Newton of $\frac{\partial^2 E}{\partial \boldsymbol{u}_i^2}\delta\boldsymbol{u}_i = -\frac{\partial E}{\partial \boldsymbol{u}_i}$ in parallel. It is known that such block-descent-like strategy suffers from slow convergence for stiff problems [Nocedal and Wright 2006].

For node i, one can rearrange all the nodal DOFs as:

$$\boldsymbol{u} = \big[\boldsymbol{u}_i^\top, \underbrace{\boldsymbol{u}_1^\top, \ldots, \boldsymbol{u}_{i-1}^\top, \boldsymbol{u}_{i+1}^\top, \ldots, \boldsymbol{u}_N^\top}_{\bar{i}}\big]^\top,$$

which allows us to partition Eq. (2) block-wisely as:

$$\begin{bmatrix} \boldsymbol{H}_{i,i} & \boldsymbol{H}_{i,\bar{i}} \\ \boldsymbol{H}_{i,\bar{i}}^\top & \boldsymbol{H}_{\bar{i},\bar{i}} \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{u}_i \\ \delta\boldsymbol{u}_{\bar{i}} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{g}_i \\ -\boldsymbol{g}_{\bar{i}} \end{bmatrix}. \tag{3}$$

JGS2 shows that the convergence of Jacobi iteration can be significantly improved (to match the Newton's convergence) by predicting how an increment at $\delta\boldsymbol{u}_i$ propagates to the rest of the model. To this end, we can set $\delta\boldsymbol{u}_i = \boldsymbol{e}_x = [1,0,0]^\top$, $\delta\boldsymbol{u}_i = \boldsymbol{e}_y = [0,1,0]^\top$, and $\delta\boldsymbol{u}_i = \boldsymbol{e}_z = [0,0,1]^\top$, and compute the corresponding responses at $\boldsymbol{u}_{\bar{i}}$. This leads to:

$$\delta\boldsymbol{u} = \boldsymbol{\phi}_i(\delta\boldsymbol{u}_i) = \begin{bmatrix} \delta\boldsymbol{u}_i \\ \delta\boldsymbol{u}_{\bar{i}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_3 \\ -\boldsymbol{H}_{\bar{i},\bar{i}}^{-1}\boldsymbol{H}_{i,\bar{i}}^\top \end{bmatrix} \delta\boldsymbol{u}_i. \tag{4}$$

Substituting Eq. (4) into the subproblem of $\arg\min_{\boldsymbol{u}_i} E(\boldsymbol{u}_i)$, the local Newton becomes:

$$\tilde{\boldsymbol{H}}_{i,i}\delta\boldsymbol{u}_i = -\tilde{\boldsymbol{g}}_i, \tag{5}$$

where $\tilde{\boldsymbol{H}}_{i,i} = \left(\frac{\partial\boldsymbol{\phi}_i}{\partial\boldsymbol{u}_i}\right)^\top \boldsymbol{H}\frac{\partial\boldsymbol{\phi}_i}{\partial\boldsymbol{u}_i}$, and $\tilde{\boldsymbol{g}}_i = \left(\frac{\partial\boldsymbol{\phi}_i}{\partial\boldsymbol{u}_i}\right)^\top \boldsymbol{g}$ denote the reduced Hessian and gradient within the perturbation subspace spanned by the Jacobi of $\boldsymbol{\phi}_i$. While $\frac{\partial\boldsymbol{\phi}_i}{\partial\boldsymbol{u}_i}$ is nonlinear and varies w.r.t. $\boldsymbol{u}$, it can be well-approximated with a co-rotated formula:

$$\frac{\partial\boldsymbol{\phi}_i}{\partial\boldsymbol{u}_i} \approx \boldsymbol{U}_i \triangleq \begin{bmatrix} \boldsymbol{I}_3 \\ -\boldsymbol{R}_{\bar{i}}\bar{\boldsymbol{H}}_{\bar{i},\bar{i}}^{-1}\bar{\boldsymbol{H}}_{i,\bar{i}}^\top\boldsymbol{R}_i \end{bmatrix}, \tag{6}$$

where $\boldsymbol{R}_{\bar{i}} \in \mathbb{R}^{3(N-1)\times 3(N-1)}$ and $\boldsymbol{R}_i \in \mathbb{R}^{3\times 3}$ are two block-diagonal matrices. Each of their 3×3 diagonal blocks represents the corresponding nodal rotation w.r.t. the rest shape, which can be efficiently computed via the polar decomposition in parallel. The notation $\bar{(\cdot)}$ suggests variables are defined in the rest shape, and $\bar{\boldsymbol{H}}_{\bar{i},\bar{i}}^{-1}\bar{\boldsymbol{H}}_{i,\bar{i}}$ is pre-computed. In other words, the nonlinearity of $\frac{\partial\boldsymbol{\phi}_i}{\partial\boldsymbol{u}_i}$ is moved to node-wise rotations, and the resulting Jacobi iterations converge nearly as effectively as Newton.

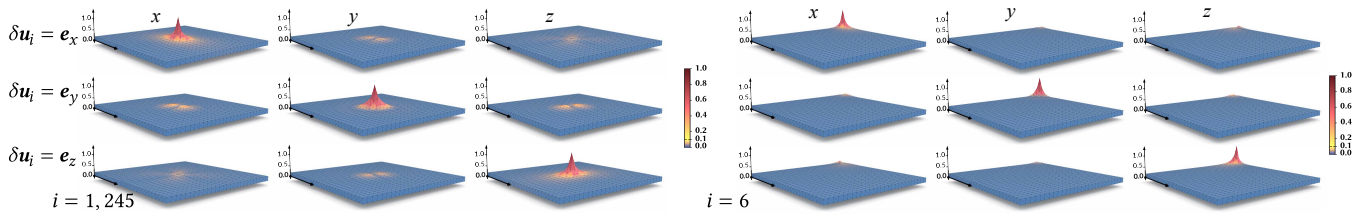


Fig. 2. **Kernel visualization on a square board.** This figure illustrates the spatial variation of the $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ for a simple square board at its rest shape. It intuitively visualizes how $\delta \boldsymbol{u}_i$ propagate cross the model when unit perturbations $\boldsymbol{e}_x$, $\boldsymbol{e}_y$, and $\boldsymbol{e}_z$ are applied at two nodes for $i = 1, 245$ (at the center) and $i = 6$ (at the corner). The x, y, and z components of $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ are plotted as the height fields. This plot supports the key argument of JGS2-GQ that $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ can be well-approximated with a low-degree polynomial locally. In fact, the local region of kernel peaks can accurately fit a five-degree polynomial with a fitting error smaller than 5%.

3.1 Cubature integration

The real challenge for JGS2 is the subspace integration of reduced force $\tilde{\boldsymbol{g}}_i$ and $\tilde{\boldsymbol{H}}_{i,i}$ in Eq. (5). In theory, each subproblem needs to traverse all the elements on the model. Thus, a JGS2 iteration performs N model-wise traversals of all the subproblems. Lan et al. [2025] proposed to use Cubature [An et al. 2008] to accelerate the runtime performance by approximating the subspace Hessian and gradient via:

$$\begin{aligned} \tilde{\boldsymbol{H}}_{i,i} &= \sum_e \left(\boldsymbol{U}_i^{e^\top} \boldsymbol{H}^e \boldsymbol{U}_i^e \right) \approx \sum_{s \in \mathbb{S}} \left(w^s \boldsymbol{U}_i^{s^\top} \boldsymbol{H}^s \boldsymbol{U}_i^s \right), \\ \tilde{\boldsymbol{g}}_i &= \sum_e \left(\boldsymbol{U}_i^{e^\top} \boldsymbol{g}^s \right) \approx \sum_{s \in \mathbb{S}} \left(w^s \boldsymbol{U}_i^{s^\top} \boldsymbol{g}^s \right), \end{aligned} \tag{7}$$

where $\boldsymbol{U}_i^e \in \mathbb{R}^{12 \times 3}$ is the element subspace matrix, which extracts pertaining rows from $\boldsymbol{U}_i$ and w^s is is the non-negative weight coefficient. The superscript s suggests the element becomes the sample set $\mathbb{S}$ of Cubature.

On the one hand, Cubature sampling substantially accelerates simulation performance since it assembles the reduced Hessian/gradient only at a sparse set of $\mathbb{S}$. On the other hand, Cubature is sensitive in JGS2 and often fails to work robustly. This issue likely arises from two factors. First, Cubature is designed for integrating subspaces spanned by low-frequency modes, assuming the training bases represent dominant global deformations. In JGS2, however, the perturbational influence from a node is often localized and high-frequency. Second, Cubature training is most effective for linear subspaces with constant basis vectors, whereas JGS2 subproblems involve nonlinear bases as defined in Eq. (6). Our key argument is that while $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ contains a locally salient signal at node i, it remains smooth enough for low-degree polynomial approximation. Fig. 2 visualizes the x, y, and z components of $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ as height fields for two nodes on a square board. They form kernel-like distributions covering the model. Nevertheless, their distributions within the local region e.g., the three-ring neighbor of node i, accurately fit a five-degree polynomial with a fitting error below 5%. This observation inspires us to opt for a training-free subspace integration strategy based on Gaussian quadrature.

4 Gaussian Quadrature

Numerical quadrature approximates definite integrals over an interval i.e., $[-1, 1]$ using a finite set of weighted samples: $\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum w_k f(x_k)$. Gaussian quadrature or GQ smartly chooses sample points and their associated weights such that an n-point rule *exactly* integrates polynomials of degree up to $2n - 1$.

Such superior accuracy and efficiency come from the connection to orthogonal polynomials. Two scalar-value functions $f_1(x)$ and $f_2(x)$ are considered orthogonal on $[-1, 1]$ if their inner product is vanished i.e., $\langle f_1, f_2 \rangle = \int_{-1}^{1} f_1(x) f_2(x)\mathrm{d}x = 0$. GQ picks sample points for an integral of $\int_{-1}^{1} f(x)\mathrm{d}x$ as the roots of *Legendre polynomials*, defined as:

$$L_n(x) = \frac{1}{2^n n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \left(x^2 - 1 \right)^n, \tag{8}$$

where n denotes the degree of the polynomial. The Legendre polynomial of degree n is orthogonal to *any polynomial* of a degree lower than n over $[-1, 1]$.

Let $P_{2n-1}(x)$ be a polynomial of degree $2n - 1$. One can apply the polynomial division to obtain:

$$P_{2n-1}(x) = Q_{n-1}(x) L_n(x) + R(x), \tag{9}$$

where $Q_{n-1}(x)$ is a polynomial of degree up to $n-1$ i.e., the quotient polynomial, and $R(x)$ is a polynomial whose degree is lower than n i.e., the remainder polynomial. The integral of $P_{2n-1}(x)$ then becomes:

$$\int_{-1}^{1} P_{2n-1}(x)\mathrm{d}x = \int_{-1}^{1} Q_{n-1}(x) L_n(x)\mathrm{d}x + \int_{-1}^{1} R(x)\mathrm{d}x. \tag{10}$$

Since $Q_{n-1}(x)$ is a lower-degree polynomial than $L_n(x)$, the orthogonality of Legendre polynomials makes the first term on the r.h.s. vanished, and the integral of $P_{2n-1}(x)$ reduces to the integral of $R(x)$.

Setting the sample points x_k as the roots of $L_n(x)$ i.e., $L_n(x_k) = 0$, the Gauss-Legendre weights w_k can be calculated via the moment-fitting method and have the following closed form:

$$w_k = \frac{2}{\left(1 - x_k^2\right) \left[L_n'(x_k) \right]^2}. \tag{11}$$

Here, w_k are uniquely determined by enforcing exactness for all polynomials of degree at most $n-1$, or mathematically:

$$\sum_{k=1}^{n} w_k P_m(x_k) = \int_{-1}^{1} P_m(x)\mathrm{d}x, \quad \forall\, m \le n-1. \tag{12}$$

Because $R(x)$ is a polynomial of a degree at most n, the n-point GQ rule integrates $R(x)$ exactly. Therefore, we have:

$$\sum_{k=1}^{n} w_k P_{2n-1}(x_k) = \int_{-1}^{1} R(x)\mathrm{d}x = \int_{-1}^{1} P_{2n-1}(x)\mathrm{d}x. \tag{13}$$

As a result, our assumption that $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ can be well-approximated by low-degree polynomials puts GQ a strong candidate for subspace integration of the reduced gradient/Hessian at each subproblem. We only need to determine the order of the integration (i.e., the degree of $P(x)$), and the optimal GQ points x_k as well as the weight coefficients w_k become readily available.

Besides the Gaussian-Legendre rule, other GQ schemes are also available, such as the Gaussian-Chebyshev and Gaussian-Hermite quadrature. These variations generalize the Gaussian-Legendre to accommodate the weighted integral of $\int_{-1}^{1} f(x)\rho(x)\mathrm{d}x$ for a prescribed density function $\rho(x)$. In our setting, however, we are interested in the reduced force integral $\sum_e \boldsymbol{U}_i^{e\top} \tilde{\boldsymbol{g}}_i$, where the distribution of $\tilde{\boldsymbol{g}}_i$ is unknown. Introducing a biased choice of $\rho(x)$ may therefore lead to a more biased approximation and eventually crash the simulation. Gaussian-Legendre quadrature corresponds to the uniform density $\rho(x) = 1$, which turns out to be the most stable and generic integration scheme for JGS2. Therefore, we will use GQ to refer to the Gaussian-Legendre rule in the rest of the paper.

5 JGS2 with Gaussian Quadrature

As discussed in Sec. 3, the key challenge in a 2nd-order Jacobi iteration is to evaluate $\tilde{\boldsymbol{g}}_i$ i.e., the reduced gradient for each subproblem i. It should be noted that $\tilde{\boldsymbol{g}}_i$ itself is a numerical approximation of the gradient integral:

$$\tilde{\boldsymbol{g}}_i \approx \int_{\Omega} \boldsymbol{f}_i(\boldsymbol{\xi})\mathrm{d}\boldsymbol{\xi}, \quad \boldsymbol{f}_i \triangleq \frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}(\boldsymbol{\xi})^{\top} \boldsymbol{g}(\boldsymbol{\xi}). \tag{14}$$

Here, $\boldsymbol{f}_i(\boldsymbol{\xi}) \in \mathbb{R}^3$ for $\boldsymbol{\xi} \in \Omega$ is the spatially varying gradient density defined on the model's volume Ω, which can be understood as the projection of the energy density's gradient $\boldsymbol{g}(\boldsymbol{\xi})$ onto the tangent manifold of $\boldsymbol{\phi}_i$. Under FEM (finite element method), this volumetric integral is approximated piece-wisely and summed up across elements, and further sparsified with Cubature samples in vanilla JGS2. Alternatively, we calculate Eq. (14) in a non-FEM way based on GQ without resorting to pre-simulation of training poses.

While GQ possesses many desirable properties for fast gradient integration, it does not apply to Eq. (14) naïvely. Dimensionality is the first incompatibility. Standard GQ numerically integrates a scalar-value function over a one-dimensional interval. In our case however, $\boldsymbol{f}_i$ is three-dimensional, and Ω represents an irregular volumetric domain. Another concern regards the polynomiality of $\boldsymbol{f}_i$. Given the uncertainty of $\boldsymbol{g}$, it is challenging to estimate the polynomial degree that always well matches $\boldsymbol{f}_i$ across Ω.

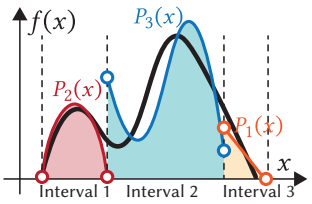


Fig. 3. **Local GQ.** The integral of a wiggly function $f(x)$ can be approximated at non-overlapping intervals with low-degree polynomials. The setup of GQ remains unchanged regardless of the specific polynomials used for the local approximation. Those local polynomials *do not* need to form a continuous global approximation of $f(x)$.

We show that those two issues can be resolved handily. It is possible to escalate the integration domain to three dimensions using the tensor product of one-dimensional GQ rules for an axis-aligned 3D box region. While finding a global polynomial to fit $\boldsymbol{f}_i$ may be tricky, $\boldsymbol{f}_i$ can be well approximated locally as the composition of a collection of low-degree polynomials. Since Eq. (14) only concerns the integral of $\boldsymbol{f}_i$, *localized polynomials do not need to be continuous across* Ω. In other words, we can use discontinuous polynomials to approximate the integration of $\boldsymbol{f}_i$ at different regions as shown in Fig. 3.

5.1 Mesh voxelization

Generalizing one-dimensional GQ to three-dimensional volumes of regular geometries, like sphere, cylinder [Canuto et al. 2006] or polyhedral [Keast 1986] is possible. Yet, the most convenient integration domain is an axis-aligned box region, which is often believed to have the best accuracy [Gerstner and Griebel 2003]. Bearing this in mind, we generate integration domains based on the voxelization of an input 3D model.

The voxelization procedure first computes the axis-aligned bounding box (AABB) of the input mesh. Given a user-specified resolution $1/h$, we build a slightly expanded bounding box to avoid floating-point ambiguity. The resulting AABB is then rasterized into a voxel grid of the size $H_x \times H_y \times H_z$. For each surface triangle, we perform a box-triangle intersection test [Akenine-Möller 2005] and generate a chunk of a face-connected voxel enclosure of this triangle. These voxels are then labeled as `flag` $= 0$, meaning they are surface voxels.

If the model's surface is watertight, the superset of all the triangle voxelization is also watertight, which partitions the AABB into the interior and the exterior. It should be noted that interior voxels and exterior voxels are not face-connected in general. In order to correctly label all the interior voxels, we perform BFS (breadth-first search) at any unlabeled voxel that resides on the surface of the AABB until all the grid's surface voxels are marked. This is because any voxel at the top, bottom, left, right, front, or back face of the grid is an exterior voxel if it has not been flagged as surface. All of its face-connected neighbors that are not yet flagged should be exterior voxels as well. By traversing all the non-flagged voxels on the surface of the AABB, we identify all the exterior voxels (`flag` $= 2$), and the remaining unlabeled voxels are the interior ones (`flag` $= 1$). We employed a grid resolution ranging from 60^3 to 200^3. A reasonable voxel size is selected to roughly match the characteristic size of the elements. Furthermore, we adaptively adjust the resolution to ensure that tetrahedra that are spatially proximal but topologically disjoint are not erroneously merged due to voxelization artifacts.

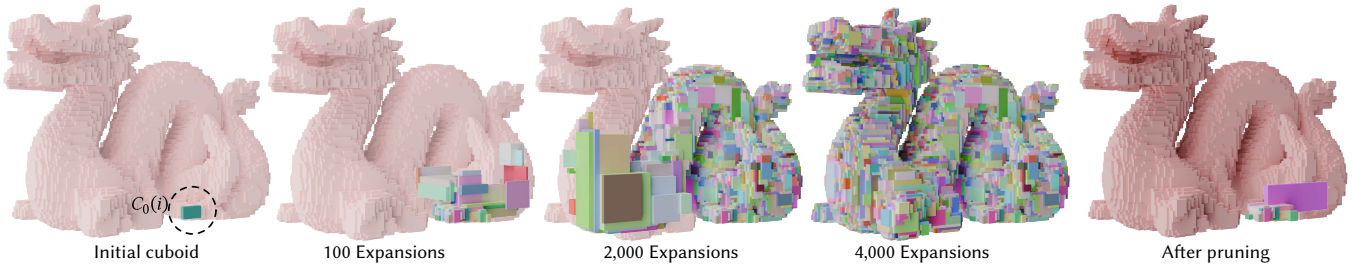


Fig. 4. **Cuboid flooding & pruning.** To facilitate GQ-based integration, we generate a collection of axis-aligned cuboids as discrete integration domains. Starting from the voxel containing node i, we iteratively expand the integration volume while maintaining its rectangular shape. Once $C_0(i)$ is formed (left-most subfigure), we evaluate its six faces and attempt to attach adjacent cuboids. This process operates on a voxelized grid until all voxels are enclosed. In this example, the domain is fully covered after 4,000 expansions. Finally, we prune nonessential cuboids to retain a sparse, efficient set of quadrature points for integration (right-most subfigure).

5.2 Cuboid generation

For the subproblem i (i.e., the i-th node), we generate a set of box-shaped integration domains, and we name each domain a *cuboid* based on the rasterized model $\mathbf{V}(\Omega)$. The node i should be contained by a certain voxel v, which forms the initial setup of the seed cuboid. The procedure is outlined in Alg. 1 — we keep performing an axis-aligned expansion of the cuboid whenever possible. This is done under the greedy choice: given the current configuration of the seed cuboid $C_0(i)$ which spans all the voxels ranging from $[\mathtt{x}_{\mathtt{min}}, \mathtt{x}_{\mathtt{max}}]$, $[\mathtt{y}_{\mathtt{min}}, \mathtt{y}_{\mathtt{max}}]$, and $[\mathtt{z}_{\mathtt{min}}, \mathtt{z}_{\mathtt{max}}]$, we pick its largest cross-section, say xy section, and check if $C_0(i)$ can be expanded via either increasing $\mathtt{z}_{\mathtt{max}}$ of decreasing $\mathtt{z}_{\mathtt{min}}$ by one. Here, x, y, z denote the integer-value voxel indices. The expansion is considered legit only when all the voxels on the slice of $\mathtt{z}_{\mathtt{min}} - 1$ (or $\mathtt{z}_{\mathtt{max}} + 1$) within $[\mathtt{x}_{\mathtt{min}}, \mathtt{x}_{\mathtt{max}}]$ and $[\mathtt{y}_{\mathtt{min}}, \mathtt{y}_{\mathtt{max}}]$ are all in $\mathbf{V}(\Omega)$ i.e., with $\mathtt{flag} = 0$ or $\mathtt{flag} = 1$. If expanding along z is not possible, we check the second-largest cross-section, and then the third, trying to expand $C_0(i)$ in a similar fashion. This procedure stops when $C_0(i)$ cannot be further expanded along any axis. In other words, $C_0(i)$ represents a maximal rectangular integration domain enclosing node i, and we label all the voxels in $C_0(i)$ with $\mathtt{flag} = 3$. The initial cuboid is generated following a greedy heuristic designed to maximize the box-shaped volume encompassing the current subproblem. This strategy focuses computational effort on the most influential region for high-quality integration (see Fig. 2). While alternative partitioning strategies exist—such as SVD-based alignment or energy-weighted distribution—the greedy choice provides an optimal balance between geometric coverage and low computational overhead. In practice, this ensures that the Gaussian Quadrature points are situated within regions of high energy density, which is critical for maintaining the convergence stability of our localized integration scheme.

After the seed cuboid is ready, more cuboids are appended to populate the remaining voxelized volume of the model $\mathbf{V}(\Omega) \setminus C_0(i)$. A stack of cuboids S is built with only $C_0(i)$ at the top. We then set the current cuboid as the top of the stack ($\mathtt{S.top()}$), and check neighboring voxels of the up face of the current cuboid. If the neighboring voxel is also on the model i.e., $\mathtt{flag} = 0$ or $\mathtt{flag} = 1$, we initialize it as a *candidate* cuboid and expand it in the same way as we did for $C_0(i)$. Out of all the candidate cuboids, the one with the largest volume is chosen to be the new cuboid $C_1(i)$, which is up-adjacent to $C_0(i)$, and we push $C_1(i)$ into S. All the voxels within $C_1(i)$ are flagged as $\mathtt{flag} = 3$. The same greedy strategy is used to append a cuboid (whenever possible) for the bottom, left, right, front, and back faces of $C_0(i)$. Since all six faces of $C_0(i)$ are now processed, we remove $C_0(i)$ out of the stack by $\mathtt{S.pop()}$ and include it into the cuboid set of subproblem i i.e., $\mathbf{C}_i \leftarrow \mathbf{C}_i \cup \{C_0(i)\}$. In this example, $C_1(i)$ becomes the new stack top and current cuboid, and we proceed with appending new cuboids to $C_1(i)$'s faces. Since voxels on the model are face-connected, this simple routine guarantees that any voxel will be included by a certain cuboid eventually. Nevertheless, we terminate it when a majority of model voxels e.g., $\geq 80\%$ are marked by $\mathtt{flag} = 3$. As shown in Fig. 4, this 80% coverage threshold is more than sufficient for pruning. Such coverage reliably captures the primary volumetric bulk and core structural topology of the model. Forcing the algorithm to cover the remaining marginal voxels would primarily generate an excessive number of drastically smaller cuboids along complex, high-curvature boundaries, thereby inflating the computational overhead while yielding diminishing returns for the pruning phase. The pseudo code of the described cuboid flooding procedure is provided in Alg. 2. Fig. 4 reports the configurations of the cuboid set for a node on the surface of the dragon model after 100, 2,000, and 4,000 expansions, respectively.

Alg. 2 always generates a large number of cuboids. Consequently, a subproblem involves a lot of GQ sampling points even if we only use the one-point rule at each cuboid. How to reduce the number of cuboids while preserving the accuracy of GQ integration becomes the key obstacle we face.

5.3 Cuboid pruning & subdividion

Alg. 2 constructs cuboids to approximate the voxelized volume using 3D rectangular boxes. This geometry-oriented construction, however, has not taken the target function $\boldsymbol{f}_i(\boldsymbol{\xi})$ into account. According to Eq. (14), $\boldsymbol{f}_i(\boldsymbol{\xi})$ is the product of $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ and $\boldsymbol{g}$. The landscape of $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ is well-understood especially when $\boldsymbol{g}$ is induced by a smooth external force e.g., gravity. As explained in Sec. 3, $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ spans a subspace describing the propagation of a deformation increment $\delta \boldsymbol{u}_i$ from node i across the model. This perturbation peaks at i and decays with distance. Consequently, a cuboid distant from i is hardly affected

Algorithm 1: Cuboid generation

Input: seed voxel v, voxelized model $\mathbf{V}$ with flags
Output: a cuboid C
// Initialize cuboid with the seed voxel
$C \leftarrow \{v\}$;
// Greedy expansion along axes
while *C can be expanded along some axis* **do**
  Find axis with largest cross-section of C (e.g., xy, yz, zx);
  foreach *direction in increasing, decreasing along the axis* **do**
    if *all new voxels on the candidate slice have* `flag` $= 0$ *or* 1 **then**
      Expand C along the axis in the direction;
      break;
Mark all voxels in C with `flag` $= 3$;
return C;

Algorithm 2: Cuboid flooding

Input: seed voxel v, voxelized model $\mathbf{V}$ with flags, coverage threshold τ (e.g., $\tau = 80\%$)
Output: a set of cuboids $\mathbf{C}$
Function `GenerateSingleCuboid`(v, $\mathbf{V}$)**:**
  return *result of Alg. 1 with inputs v and $\mathbf{V}$*
// Main algorithm body begins here
Initialize: empty stack S, cuboid set $\mathbf{C} \leftarrow \emptyset$, coverage $\leftarrow 0$;
// Generate the initial seed cuboid using Alg. 1
$C_0 \leftarrow$ `GenerateSingleCuboid`(v, $\mathbf{V}$);
$\mathbf{C} \leftarrow \mathbf{C} \cup \{C_0\}$;
S.push(C_0);
Update coverage;
// Flooding process to populate the model with cuboids
while *S is not empty and coverage $< \tau$* **do**
  current $\leftarrow S$.top();
  foreach *face in top, bottom, left, right, front, back* **do**
    candidates $\leftarrow \emptyset$;
    foreach *voxel v adjacent to current on face* **do**
      if v.`flag` $= 0$ *or* 1 **then**
        // Call Alg. 1 to generate a new cuboid from candidate voxel
        cand $\leftarrow$ `GenerateSingleCuboid`(v, $\mathbf{V}$);
        candidates $\leftarrow$ candidates $\cup$ {cand};
    if *candidates $\neq \emptyset$* **then**
      bestCand $\leftarrow$ candidate with largest volume from candidates;
      $\mathbf{C} \leftarrow \mathbf{C} \cup$ {bestCand};
      S.push(bestCand);
      Update coverage;
      if *coverage $\geq \tau$* **then**
        break from while loop;
  S.pop();
  // Remove current cuboid after processing all faces
return $\mathbf{C}$;

by $\delta \boldsymbol{u}_i$. Intuitively, we want to favor cuboids located near node i. A more grounded way is to quantify the *influence* of $\delta \boldsymbol{u}_i$ based on the shape of $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$.

5.4 Cuboid pruning

Since $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ can be well-approximated with a co-rotated formula of Eq. (6), we estimate the influence of $\delta \boldsymbol{u}_i$ at each node j, based on the rest-shape $\bar{\boldsymbol{U}}_i$ defined as:

$$\bar{\boldsymbol{U}}_i = \begin{bmatrix} \boldsymbol{I}_3 \\ -\bar{\boldsymbol{H}}_{\bar{i},\bar{i}}^{-1} \bar{\boldsymbol{H}}_{i,\bar{i}}^{\top} \end{bmatrix}. \tag{15}$$

Calculating $\bar{\boldsymbol{U}}_i$ does not introduce additional overheads to the simulation pipeline because $\bar{\boldsymbol{H}}_{\bar{i},\bar{i}}^{-1} \bar{\boldsymbol{H}}_{i,\bar{i}}^{\top}$ is already pre-computed at each subproblem.

$\bar{\boldsymbol{U}}_i \in \mathbb{R}^{3N \times 3}$ can be viewed as a tensor field, which assigns each node j a 3×3 matrix i.e., $\bar{\boldsymbol{U}}_{i,j}$ representing its *sensitivity* of displacement to the initial perturbation $\boldsymbol{I}_3$ at node i. A standard metric to convert this tensor-valued response into a positive scalar measuring the *magnitude of the influence* is the Frobenius norm:

$$\omega_j = \sqrt{\mathrm{tr}\left(\bar{\boldsymbol{U}}_{i,j}^{\top} \bar{\boldsymbol{U}}_{i,j}\right)}, \tag{16}$$

which maps $\bar{\boldsymbol{U}}_i$ to $\omega_i \in \mathbb{R}_+^N$.

ω_i offers an objective measure of the influence from a unit perturbation applied at node i. Out of it, we assign each subproblem i an average influence density defined as:

$$\bar{\rho}_i = \frac{1}{|\Omega|} \int_{\Omega} \omega_i(\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\xi}, \tag{17}$$

where $\omega_i(\boldsymbol{\xi})$ can be easily obtained by barycentric or trilinear interpolation of the discretized ω_i. Similarly, we can also estimate the influence density $\rho(C_\ell(i))$ for the ℓ-th cuboid associated with the subproblem i. Let $l_\ell^x(i)$, $l_\ell^y(i)$, and $l_\ell^z(i)$ be the three edges of $C_\ell(i)$, and $V_\ell(i)$ be its volume such that $V_\ell(i) = l_\ell^x(i) l_\ell^y(i) l_\ell^z(i)$. $\rho(C_\ell(i))$ is the volume-normalized sum of ω_i over all the voxels within $C_\ell(i)$. If $\rho(C_\ell(i)) \leq 0.05\bar{\rho}_i$, $C_\ell(i)$ is considered less important for subspace integration and discarded. As shown in the right-most subfigure of Fig. 4, such influence density-based pruning excludes most cuboids remote from i, while the integration accuracy is barely impacted.

5.4.1 Cuboid subdivision. Another aspect of enabling both accuracy and sparsity in GQ integration is the cuboid size. A larger cuboid covers more volume, but the polynomiality of $\boldsymbol{f}_i$ within becomes compromised, and a higher-order integration rule is likely necessary. Conversely, a smaller cuboid allows far fewer GQ points for a high-quality integration, but only for a limited volume.

The optimal trade-off can also be determined based on the influence density. Since the influence $\omega_i(\boldsymbol{\xi})$ diminishes as one moves away from node i, the integration accuracy naturally hinges primarily on its immediate vicinity. Alg. 1, however, only seeks the largest possible rectangular domain $C_0(i)$ housing i. In an extreme case, for instance when Ω is itself a rectangular box, $C_0(i)$ for any subproblem would cover the entire Ω. This is equivalent to replacing the original model with a single (high-order) hexahedral element.

Based on this analysis, we subdivide $C_0(i)$ into three smaller cuboids along its longest edge if $\rho(C_0(i)) \leq 0.7\bar{\rho}_i$. This subdivision

ensures the assignment of GQ points at $C_0(i)$ is more effective for improving integration accuracy.

6 GQ Integration

With the integration domains established, we apply the 3D GQ rule to each cuboid. Unlike Cubature, the optimal quadrature points and weights are analytically determined as long as an integration order is chosen, making JGS2-GQ training-free.

6.1 Cubic Gaussian quadrature

The most convenient way to lift a one-dimensional GQ to 3D is via the tensor-product construction. Let $P(x)$ denote a polynomial integrand and $\{(x_k, w_k)\}_{k=1}^{n}$ be the n-point GQ rule associated with the Legendre polynomial $L_n(x)$.

Consider the integration of a polynomial $P(x, y, z)$ over a rectangular domain i.e., a cuboid $C = [a_x, b_x] \times [a_y, b_y] \times [a_z, b_z]$. By applying an affine change of variables in each direction, the integral over C can be mapped to the reference cube $[-1, 1]^3$ as:

$$\int_C P(x, y, z)\, \mathrm{d}x\, \mathrm{d}y\, \mathrm{d}z = J \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \bar{P}(x, y, z)\, \mathrm{d}x\mathrm{d}y\mathrm{d}z, \tag{18}$$

where $\bar{P}$ denotes the integrand expressed in the reference coordinates, and we have

$$J = \frac{(b_x - a_x)(b_y - a_y)(b_z - a_z)}{8}, \tag{19}$$

which is the Jacobian determinant of the domain transformation.

The 3D GQ rule is then obtained as the tensor product of the corresponding 1D rules,

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \bar{P}(x, y, z)\mathrm{d}x\mathrm{d}y\mathrm{d}z \approx \sum_{k_x=1}^{n_x}\sum_{k_y=1}^{n_y}\sum_{k_z=1}^{n_z} w_{k_x} w_{k_y} w_{k_z}\, \bar{P}(x_{k_x}, y_{k_y}, z_{k_z}). \tag{20}$$

Similar to its 1D counterpart, the resulting 3D quadrature rule integrates exactly all polynomials whose degree in each coordinate direction does not exceed $2n - 1$.

$\boldsymbol{f}_i(\boldsymbol{\xi})$ represents a vector field of the reduced gradient with coupled x, y, and z components. Fortunately, GQ depends only on the degree of the underlying polynomial approximation. If the x-component $f_{i,x}(\boldsymbol{\xi})$ is well-approximated by an n-degree polynomial $P_n^x(x)$, it is reasonable to assume that $f_{i,y}(\boldsymbol{\xi})$ and $f_{i,z}(\boldsymbol{\xi})$ are also well-approximated by n-degree polynomials $P_n^y(y)$ and $P_n^z(z)$. Although these polynomials may take drastically different forms, GQ yields accurate integration as long as their orders are properly estimated. The process is agnostic to the specific coupling between the components.

6.2 Order policy

The final piece of the missing puzzle is the integration order, i.e., how many GQ points we should use at each cuboid. Pushing the accuracy requires higher-order integration with more GQ points, while an efficient runtime simulation favors fewer GQ points, considering that each subproblem is executed on a single CUDA thread. Given the nature of tensor expansion, we focus on GQ orders of one, two, and three. A higher-order GQ rule is less profitable in practice, and subdividing the integration domain with multiple lower-point rules is more effective and efficient.

While all cuboids are rectangular, their aspect ratios can vary significantly, ranging from near-equilateral volumes to compressed planes or elongated sticks. Standard expansion of 1D GQ to 3D based on the reference cube $[-1, 1]^3$ typically implies the same integration orders in all three directions. However, when a cuboid becomes nearly collinear or planar, we should decouple these orders by employing a higher-order rule along the dominant dimensions and a lower-order rule along the compressed ones. This geometric information is extracted by evaluating the ratios between $\{l_\ell^x(i), l_\ell^y(i), l_\ell^z(i)\}$ and their maximum value $\max\{l_\ell^x(i), l_\ell^y(i), l_\ell^z(i)\}$. The influence density $\rho(C_\ell(i))$ remains an informative checkpoint. We previously utilized this metric to exclude less significant cuboids (see Sec. 5.3). A low $\rho(C_\ell(i))$ suggests that it is safe to downgrade the corresponding GQ integration to a lower order. In addition to density, the total influence encompassed by $C_\ell(i)$ should be considered as well. A large integration domain remains worthy of sampling even if its density is low, as its cumulative contribution to the global response can still be substantial.

Putting those together, we design an order score for each dimension of $C_\ell(i)$ denoted as $S_\ell^x(i)$, $S_\ell^y(i)$, and $S_\ell^z(i)$ such that:

$$S_\ell^{x,y,z}(i) = \alpha_1 \frac{l_\ell^{x,y,z}(i)}{\max\{l_\ell^x(i), l_\ell^y(i), l_\ell^z(i)\}} + \alpha_2 \frac{\rho(C_\ell(i))}{\max\{\rho(C_\ell(i))\}} + \alpha_3 \frac{\rho(C_\ell(i))V_\ell(i)}{\max\{\rho(C_\ell(i))V_\ell(i)\}}. \tag{21}$$

This scoring system incorporates the three factors previously discussed: geometric shape, influence density, and total influence. Each component is normalized to the range $[0, 1]$, with weights empirically set to $\alpha_1 = 0.4$, $\alpha_2 = 0.3$, and $\alpha_3 = 0.3$. Finally, the order score $S_\ell^{x,y,z}(i)$ determines the GQ degree along each dimension via a staggered mapping:

$$\mathrm{Deg}_\ell^{x,y,z}(i) = \begin{cases} 1, & \text{if } S_\ell^{x,y,z}(i) < 1/3, \\ 2, & \text{if } 1/3 \le S_\ell^{x,y,z}(i) < 2/3, \\ 3, & \text{if } S_\ell^{x,y,z}(i) \ge 2/3. \end{cases} \tag{22}$$

6.3 GQ (integrand) or Cubature (integral)

While both GQ and Cubature [An et al. 2008] are effective numerical integration schemes, we would like to emphasize that GQ is *not* the best solution for reduced simulations in general. We apply 3D GQ here specifically because each subproblem is rank-three. At this dimension, it is reasonable to assume a low-degree polynomial well fits the target function $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ over a wide region e.g., a cuboid. In other words, JGS2-GQ aims to approximate the *integrand* for the reduced integration. On the other hand, a typical model reduction employs tens or hundreds of subspace bases. At this rank, reduced gradient and Hessian could lose the low-degree polynomiality over a coarser integration domain. Therefore, GQ becomes less effective (if not useless). Cubature directly learns the *integral*, rather than the integrand, based on sampled deformations. If the simulation results are covered by the training data, Cubature is highly effective since the training at the integral-level hides the complexity of the

integrand, allowing even fewer sampling points. As with other data-driven methods, its accuracy may be undermined for previously unseen states. Nevertheless, Cubature remains the most practical and effective approach for general reduced simulations.

7 JGS2-GQ for Contacts

The presence of collision, contact, and friction introduces an added layer of challenge. A fully implicit collision alters the system Hessian and couples the DOFs involved in colliding primitives. A naïve treatment is to augment the local Hessian in subproblem i such that $\tilde{\boldsymbol{H}}_{i,i} \leftarrow \tilde{\boldsymbol{H}}_{i,i} + \frac{\partial^2 C}{\partial \boldsymbol{u}_i^2}$ for some contact penalties C. However, this strategy fails to account for the body-level coupling induced by contact, leading to degraded convergence when models remain in contact for prolonged periods. To address this, we propose an efficient improvement by modifying the global $\bar{\boldsymbol{U}}_i$, and extend this idea to handle the friction. Our key observation is: the coupling between bodies is enabled through the 12×12 collision stencil, while the within-body perturbation subspace is largely unchanged.

7.1 Contact-modified subspace

Without loss of generality, we consider two objects A and B colliding with each other via a node-triangle pair, where the colliding node is on body A, and the triangle is on body B. In this case, $\boldsymbol{u} = [\boldsymbol{u}^{A^\top}, \boldsymbol{u}^{B^\top}]^\top$ includes all the DOFs on both bodies. We denote the global indices of the colliding node on A as a, and three triangle nodes on B as b_1, b_2, and b_3.

When A and B are separate, $\boldsymbol{H}$ is block diagonal, and all the DOFs on A and B are decoupled. In this case, $\boldsymbol{\phi}_a(\delta\boldsymbol{u}_a)$ vanishes on B. When A and B are in contact, we note that the perturbation of $\delta\boldsymbol{u}_a$ also produces $\delta\boldsymbol{u}^B$ if the perturbation tends to reduce the node-triangle distance. Therefore, $\boldsymbol{\phi}_a(\delta\boldsymbol{u}_a)$ consists of two parts. $\boldsymbol{\phi}_a^A$ represents the propagation of the nonlinear elasticity on A, which can still be calculated via Eq. (6). On the other hand, $\boldsymbol{\phi}_a^B$ embodies the coupling through the collision. We model such contact-based coupling as a two-stage perturbation propagation: how the perturbation applied at a reaches b_1, b_2, and b_3 via the contact model, and how the perturbation further distributes over B from b_1, b_2, and b_3.

Let $C(\boldsymbol{u}_a, \boldsymbol{u}_{b_1}, \boldsymbol{u}_{b_2}, \boldsymbol{u}_{b_3})$ be the contact potential associated with this node-triangle pair, such as IPC [Li et al. 2020], and $\boldsymbol{H}^C \in \mathbb{R}^{12\times12}$ be the corresponding collision stencil. We partition $\boldsymbol{H}^C$ into four blocks:

$$\boldsymbol{H}^C = \begin{bmatrix} \boldsymbol{H}^C_{a,a} & \boldsymbol{H}^C_{a,b} \\ \boldsymbol{H}^{C^\top}_{a,b} & \boldsymbol{H}^C_{b,b} \end{bmatrix},$$

where $\boldsymbol{H}^C_{b,b}$ is a 9×9 sub-matrix corresponding to b_1, b_2, b_3 on body B. $\delta\boldsymbol{u}_a$-triggered perturbation arriving at the triangle is computed via:

$$\delta\boldsymbol{u}_b = \begin{bmatrix} \delta\boldsymbol{u}_{b_1} \\ \delta\boldsymbol{u}_{b_2} \\ \delta\boldsymbol{u}_{b_3} \end{bmatrix} = \left(-\boldsymbol{H}^{C^{-1}}_{b,b}\boldsymbol{H}^{C^\top}_{a,b}\right)\delta\boldsymbol{u}_a. \tag{23}$$

Since $\boldsymbol{H}^C_{b,b}$ is a small-size matrix, solving Eq. (23) is efficient and can be parallelized at all the contact pairs.

Technically, $\delta\boldsymbol{u}_b$ should also accommodate the elasticity response from B, which would require solving the global system in Eq. (3) with time-varying collision Hessian. It is clearly prohibitive and cannot be pre-computed. We opt for a simplified one-way collision-elasticity coupling to facilitate the construction of the perturbation subspace. We argue that this treatment is reasonable given the fact that collision stiffness is typically orders of magnitude stronger than elasticity. Therefore, the collision potential is insensitive to the body's elastic deformation. For instance, in IPC, the sharply increasing barrier prevents the node-triangle distance from being significantly reduced by elasticity alone. In practice, the colliding pair can even be treated as rigid, simplifying Eq. (23) to:

$$\delta\boldsymbol{u}_b = \left(-\boldsymbol{H}^{C^{-1}}_{b,b}\bar{\boldsymbol{H}}^{C^\top}_{a,b}\right)\delta\boldsymbol{u}_a \approx \left(\mathbf{1}_{3\times1}\otimes\left(\boldsymbol{n}\boldsymbol{n}^\top\right)\right)\delta\boldsymbol{u}_a, \tag{24}$$

which passes the normal perturbation from a to b_1, b_2, and b_3. Here, $\boldsymbol{n}$ is the contact normal. In other words, we have $\delta\boldsymbol{u}_{b_1} = \delta\boldsymbol{u}_{b_2} = \delta\boldsymbol{u}_{b_3} = (\boldsymbol{n}\boldsymbol{n}^\top)\delta\boldsymbol{u}_a$.

After $\delta\boldsymbol{u}_b$ is obtained, $\boldsymbol{\phi}_a^B$ becomes the superposition of three body-wise perturbations induced by $\boldsymbol{u}_{b_1}$, $\boldsymbol{u}_{b_2}$, and $\boldsymbol{u}_{b_3}$, and we have:

$$\begin{aligned}\frac{\partial\boldsymbol{\phi}_a^B}{\partial\boldsymbol{u}_a} &= \frac{\partial\boldsymbol{\phi}_a^B}{\partial\boldsymbol{u}_{b_1}}\frac{\partial\boldsymbol{u}_{b_1}}{\partial\boldsymbol{u}_a} + \frac{\partial\boldsymbol{\phi}_a^B}{\partial\boldsymbol{u}_{b_2}}\frac{\partial\boldsymbol{u}_{b_2}}{\partial\boldsymbol{u}_a} + \frac{\partial\boldsymbol{\phi}_a^B}{\partial\boldsymbol{u}_{b_3}}\frac{\partial\boldsymbol{u}_{b_3}}{\partial\boldsymbol{u}_a} \\ &= \frac{\partial\boldsymbol{\phi}^B_{b_1}}{\partial\boldsymbol{u}_{b_1}}\frac{\partial\boldsymbol{u}_{b_1}}{\partial\boldsymbol{u}_a} + \frac{\partial\boldsymbol{\phi}^B_{b_2}}{\partial\boldsymbol{u}_{b_2}}\frac{\partial\boldsymbol{u}_{b_2}}{\partial\boldsymbol{u}_a} + \frac{\partial\boldsymbol{\phi}^B_{b_3}}{\partial\boldsymbol{u}_{b_3}}\frac{\partial\boldsymbol{u}_{b_3}}{\partial\boldsymbol{u}_a}.\end{aligned} \tag{25}$$

Here, $\frac{\partial\boldsymbol{\phi}^B_{b_1}}{\partial\boldsymbol{u}_{b_1}}$, $\frac{\partial\boldsymbol{\phi}^B_{b_2}}{\partial\boldsymbol{u}_{b_2}}$, and $\frac{\partial\boldsymbol{\phi}^B_{b_3}}{\partial\boldsymbol{u}_{b_3}}$ can be calculated via Eq. (6) efficiently using the co-rotated formula. Meanwhile, $\frac{\partial\boldsymbol{u}_{b_1}}{\partial\boldsymbol{u}_a}$, $\frac{\partial\boldsymbol{u}_{b_2}}{\partial\boldsymbol{u}_a}$, and $\frac{\partial\boldsymbol{u}_{b_3}}{\partial\boldsymbol{u}_a}$ are the top, middle, and bottom 3×3 blocks of $-\boldsymbol{H}^{C^{-1}}_{b,b}\boldsymbol{H}^{C^\top}_{a,b}$ or simply $\boldsymbol{n}\boldsymbol{n}^\top$ if Eq. (24) is used.

7.2 Frictional contacts

When collisions and contacts are frictional, we cannot formulate the entire system as a potential-based minimization problem, and JGS2 may not be directly applicable. However, it is possible to approximate the friction as a lagged penalty e.g., in [Li et al. 2020].

During the construction of subspace for each subproblem, the Coulomb friction can be conveniently modeled as a perturbation amendment. Taking the same node-triangle collision as an example, we first estimate a contact force increment of this collision pair based on the nodal velocity from the previous time step:

$$\delta\boldsymbol{f} = \left[\delta\boldsymbol{f}_a^\top, \delta\boldsymbol{f}_{b_1}^\top, \delta\boldsymbol{f}_{b_2}^\top, \delta\boldsymbol{f}_{b_3}^\top\right]^\top = \boldsymbol{H}^C\left[\hat{\boldsymbol{u}}_a^\top, \hat{\boldsymbol{u}}_{b_1}^\top, \hat{\boldsymbol{u}}_{b_2}^\top, \hat{\boldsymbol{u}}_{b_3}^\top\right]^\top. \tag{26}$$

We keep our focus on node a as the current subproblem. If there exists relative sliding velocity among a, and b_1, b_2, b_3 at the collision tangent, we modify $\frac{\partial\boldsymbol{u}_b}{\partial\boldsymbol{u}_a}$ to reflect the Coulomb model. Specifically,

$$\frac{\partial\boldsymbol{u}_b}{\partial\boldsymbol{u}_a} = \mathbf{1}_{3\times1}\otimes\left(\boldsymbol{n}\boldsymbol{n}^\top + \frac{\boldsymbol{u}_\parallel\delta\boldsymbol{u}_a^\top}{\delta\boldsymbol{u}_a^\top\delta\boldsymbol{u}_a}\right), \tag{27}$$

where

$$\boldsymbol{u}_\parallel \triangleq \mu\left\|\delta\boldsymbol{f}_a - \frac{1}{3}\left(\delta\boldsymbol{f}_{b_1} + \delta\boldsymbol{f}_{b_2} + \delta\boldsymbol{f}_{b_3}\right)\right\|\frac{\left(\boldsymbol{I}_3 - \boldsymbol{n}\boldsymbol{n}^\top\right)\delta\boldsymbol{u}_a}{\left\|\left(\boldsymbol{I}_3 - \boldsymbol{n}\boldsymbol{n}^\top\right)\delta\boldsymbol{u}_a\right\|}. \tag{28}$$

In other words, we include a tangential perturbation to predict the Coulomb effect, which induces *stickiness* on the contact tangent such that $\delta\boldsymbol{u}_a$ and $\delta\boldsymbol{u}_{b_{1,2,3}}$ are coupled. If the contact plane is friction-free, $\delta\boldsymbol{u}_a$ does not affect $b_{1,2,3}$ tangentially, and thus $\boldsymbol{u}_\parallel = \mathbf{0}$. Otherwise,

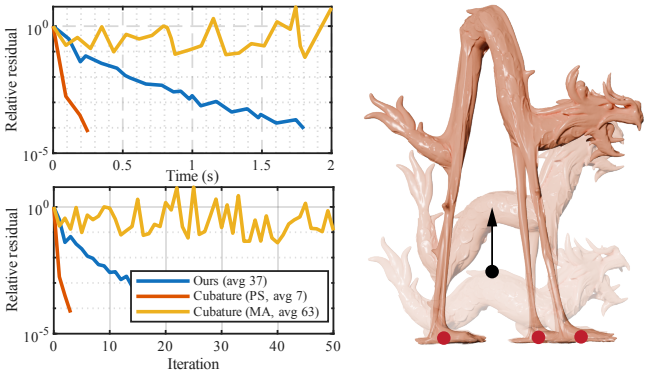


Fig. 5. **Comparison with JGS2-Cubature.** Convergence comparison on an Asian dragon model (1M DOFs) under sharp local deformations. While Cubature with scene-specific training (i.e., pre-simulation or PS) matches Newton-like speed, it fails to converge when trained on global modes using modal analysis (MA). In contrast, our training-free JGS2-GQ consistently achieves reliable convergence without any prior data.

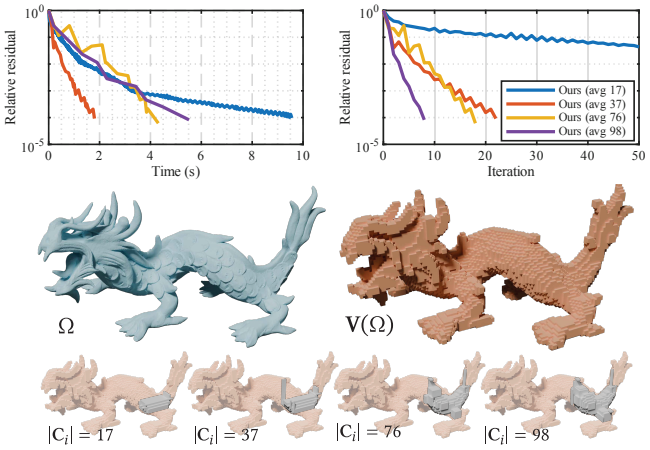


Fig. 6. **Integration vs convergence.** We study the relation between GQ integration accuracy and simulation performance in the same simulation as in Fig. 5. As the number of GQ points increases, the iteration count decreases, and the convergence improves. Such improvement plateaus when the total number of GQ points goes beyond 98. When evaluating the error reduction speed (i.e., convergence rate per unit of time), a sparser set of quadrature points provides higher computational efficiency. We also visualize the corresponding configuration of the cuboid set for the mesh node at the leg of the dragon.

$\delta \boldsymbol{u}_a$ imposes a tangential perturbation, the magnitude of which is estimated based on the Coulomb frictional coefficient μ.

We emphasize that Eqs. (24) and (27) are not intended to solve the unknown displacement vector $\boldsymbol{u}$. Rather, they are used to build an informative subspace characterizing how a small perturbation applied at i propagates to other DOFs in the system within the current tangent space. This information makes each subproblem globally aware, ensuring that local optimization is unlikely to overshoot. While the variational optimization in Eq. (1) remains unchanged, a high-quality subspace promotes better convergence of the Jacobi iterations, matching the rate of Newton's method.

7.3 Adaptive GQ order

Along with collisions and contacts come repulsive forces from collision penalties, such as IPC barriers. These penalties generate strong, high-frequency, and localized gradients of $\boldsymbol{g}$ that alter the landscape of $\boldsymbol{f}_i(\xi)$. To ensure these sharp signals are not overlooked, we *share cuboids across different subproblems* and adjust the integration order on-the-fly when necessary.

Normally, the set of cuboids and their quadrature points are generated specifically for the subproblem i, to cover the most salient features of $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$. Suppose an abrupt gradient $\boldsymbol{g}$ occurs at node j ($j \neq i$) due to a high-speed collision or sharp external force. If j falls within a cuboid $C_\ell(i) \in \mathbf{C}_i$, we simply escalate its integration order. This adjustment is efficient because 3D GQ points and weights are available in closed-form for any given order. Switching from a one-point rule to a two-point or a three-point rule, for example, requires no additional pre-computation. However, if j is not enclosed by any existing cuboids for node i, which can occur since $\mathbf{C}_i$ only encompasses sparse volumes of Ω (e.g., see Fig. 4), we leverage the fact that j is always contained by its own $C_0(j)$. We then expand the cuboid set of i such that $\mathbf{C}_i \leftarrow \mathbf{C}_i \cup \{C_0(j)\}$. This flexibility allows GQ-based integration to adaptively adjust the domains and orders to capture dominant variations in the field.

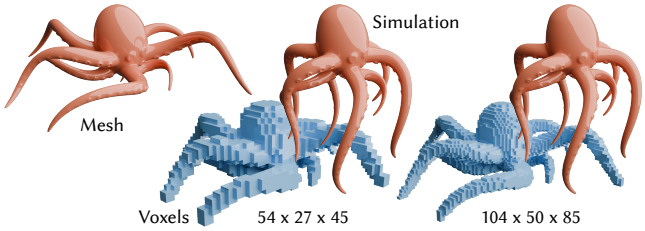


Fig. 7. **Ablation study on different voxelizations.** Evaluation of the octopus model under varying resolutions. The results show that with a consistent average of 33 quadrature points, the overall outcomes and computational costs remain stable and comparable across different voxelization levels.

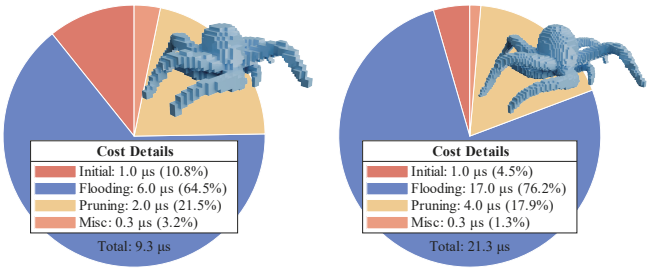


Fig. 8. **Cuboid generation cost breakdown.** This figure compares the average computational overhead for each vertex across two voxelization resolutions (left: $54 \times 27 \times 45$, right: $104 \times 50 \times 85$). In both cases, the flooding stage constitutes the primary bottleneck, accounting for approximately 65–76% of the total generation time, while initial selection and pruning remain relatively consistent.

Table 1. **Statistics for experiments.** This table details the total numbers of tetrahedra (#tets), Degrees of Freedom (#DOFs), and surface triangles (#tris). We employ Neo-Hookean [Heyliger 2008] as the constitutive model for deformable bodies, while using BW98 [Baraff and Witkin 1998] with quadratic bending [Bergou et al. 2006] for cloth simulation. Key simulation parameters such as time step (Δt), Young's Modulus (E), Poisson's ratio (ν), friction coefficient (μ) the average number of quadrature points (N) are provided. Additionally, the table includes the average CCD time per step, the average number of Newton iterations per step and the average solver time per step. We use the convergence condition of $\|\Delta \boldsymbol{x}\| = 10^{-3}$ for the experiments listed below.

Example	#tets	#DOFs	#tris	Δt (s)	E (Pa)	ν	μ	N	CCD time (s)	#iters	solver time (ms)
Puffer balls	3.8M	5.4M	3.6M	1/100	1e7/5e5	0.4	0.1	34	6.7	43	742
Brush armadillos	1.7M	1.7M	1.2M	1/150	1e9/1e7/5e5	0.4	0.1	49	7.1	36	118
Brushes x140	7.4M	9M	6M	1/100	1e9/1e7	0.4	0.1	54	18.6	55	1,146
Staircase ($\mu = 0$)	582K	371K	490K	1/30	1e7	0.45	0	33	0.9	47	121
Staircase ($\mu = 0.5$)	582K	371K	490K	1/30	1e7	0.45	0.5	33	1.7	56	215
Twisted armadillos	157K	236K	233K	1/30	1e7/5e5/1e9	0.4	0.1	26	3.2	51	746
Armadillos bowl (E = 1e6 Pa)	799K	721K	480K	1/30	1e6	0.45	0.1	47	1.1	39	127
Armadillos bowl (E = 5e5 Pa)	799K	721K	480K	1/30	5e5	0.45	0.1	47	1.3	56	142
Shooting armadillos	16K	229K	479K	1/30	1e7/1e9	0.4	0.3	25	2.7	37	181
iPhone pocket	1.2M	1.5M	987K	1/100	1e8/1e9	0.4	0	41	9.2	94	816
Multilayer cloths	-	706K	480K	1/30	-	-	0.1	12	12.2	62	387
A puffer ball	190K	174K	87K	1/100	1e6	0.4	0.3	34	2.8	22	125

8 Experimental Results

We implemented the proposed JGS2-GQ framework on a desktop computer with an `intel i7-12700` CPU (for computing $\bar{\boldsymbol{U}}_i$) and an `Nvidia 3090 RTX` GPU (for runtime simulation). Tab. 1 provides detailed experimental setups and timing statistics. Please also refer to the supplementary video for more animation results. Given that our experiments often involve stiff, extensive, and high-speed collisions among complex deformable objects, we employ line search filtering based on CCD (continuous collision detection) as described in [Li et al. 2020] to ensure the penetration-free state after each iteration. Consequently, repetitive CCD invocation becomes the primary computational bottleneck in our implementation. If the strict penetration-free guarantee is lifted and replaced by soft collision handling methods, such as those based on penetration depth [Chen et al. 2023], the overall performance can be further improved by at least one order.

8.1 Ablation on Cuboid Generation & Cost Breakdown

Cuboid generation follows three stages: initial selection, flooding, and pruning. Since voxelization resolution determines the input for all subsequent steps, we performed an ablation study to measure its impact.Fig. 7 illustrates two models generated at resolutions of $54 \times 27 \times 45$ and $104 \times 50 \times 85$. Both configurations yield an average of 33 quadrature points and exhibit similar performance (75ms/9 iterations vs. 81ms/10 iterations), suggesting that resolution has a negligible effect on efficiency. As shown in the cost breakdown (Fig. 8), flooding remains the primary bottleneck, consuming roughly 70% of the total computation time.

8.2 Comparison with JGS2 using Cubature

In the first experiment, we compare two integration schemes i.e., Cubature and GQ, within the framework of JGS2 [Lan et al. 2025]. Being a data-driven method, Cubature sampling points and weights are optimized via NNLS iteratively. As discussed, the efficacy of Cubature training depends on the quality of the training set. Specifically, Cubature is most effective if the training set spans the simulation

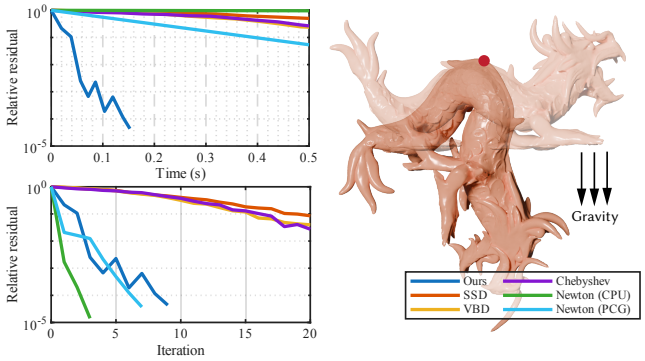


Fig. 9. **Comparison with GPU block descent-based methods.** We compare our method against state-of-the-art block descent-based GPU solvers, including VBD [Chen et al. 2024b], SSD [Lan et al. 2023], and Chebyshev Jacobi [Wang and Yang 2016]. We pin the back of the dragon model and simulate its gravitational deformation without collision resolution. Because existing methods rely on local relaxation and neighbor-to-neighbor information propagation, they suffer from inefficient convergence and numerical overshooting on high-resolution meshes. In contrast, JGS2-GQ incorporates global context through $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$, enabling significantly faster error reduction and superior stability across various complex deformation scenarios.

results. When the training poses diverge from the simulation, we observe a substantial drop in the convergence rate. That said, the Cubature-trained model extrapolates poorly, leading to inaccurate force/Hessian approximations.

Fig. 5 provides a side-by-side comparison illustrating this behavior. We fix the feet of an Asian dragon model (1M DOFs) and apply sharp external forces to its back and head, triggering large local deformations. This scene is simulated using our method and vanilla JGS2. For Cubature, we evaluate two training sets. The first set is obtained by pre-simulating the scene with Newton's method and extracting the first 30 principal components via PCA (principal component analysis). With this highly relevant training set (labeled

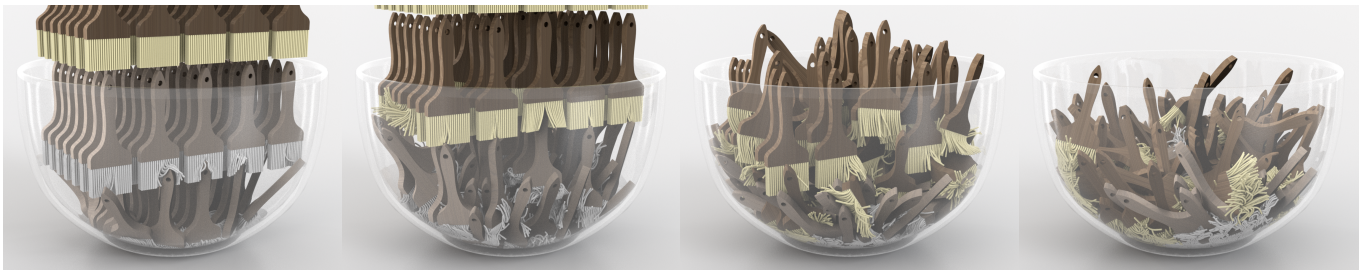


Fig. 10. **A lot of brushes.** There are 140 brushes dropping into the bowl under gravity. The yellow bristles are 20× stiffer ($E = 10^7\ Pa$) than the grey ones ($E = 5 \times 10^5\ Pa$), while the handles are nearly rigid (wooden, $E = 10^9\ Pa$). Such a heterogeneous system is beyond the capability of regular block descent-based methods like VBD. JGS2-GQ offers better robustness than the vanilla JGS2, and it remains highly efficient without training. In this example, all the collisions are resolved at each time step using IPC, and our method is 5× faster than GPU-IPC [Guo et al. 2024], and 100× faster than PD-IPC [Lan et al. 2022b]. The convergence comparison is reported in Fig. 11.

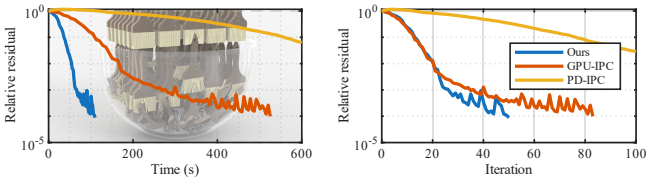


Fig. 11. **Comparison with GPU-based IPC solvers.** We compare our method with GPU-based IPC solvers. When massive collisions are present, IPC-based collision resolution is often considered the default option to avoid interpenetration. Our method outperforms CPU-based Newton-Krylov methods as we use optimized the vector inner products that typically bottleneck Krylov iterations. The plots correspond to a representative frame from the experiment in Fig. 10. Even in the scenario with extreme stiffness disparity, JGS2-GQ is 5× faster than GPU-IPC [Guo et al. 2024] and over 100× faster than PD-IPC [Lan et al. 2022b] as the latter struggles to converge under high stiffness.

as PS), Cubature effectiveness is maximized — only 7 integration points allow JGS2 to achieve a convergence rate nearly identical to Newton's method. At the other extreme, we generate 30 training poses using modal analysis (MA) [Pentland and Williams 1989] to extract low-frequency eigenvectors. Since these global deformations differ significantly from the actual local response, the simulation fails to converge, even though 63 sampling points are used per node and the training error is low. Consequently, our method achieves over 84× speedup compared to JGS2-Cubature (MA), demonstrating its robustness and training-free advantage.

Our method resides between these two extremes. The cuboids for each subproblem consistently capture the dominant information of the kernel function $\frac{\partial \phi_i}{\partial u_i}$. Consequently, JGS2-GQ offers reliable performance without relying on any prior simulation data. In this example, our method uses an average of 37 quadrature points. While it is slightly outperformed by the perfectly optimized Cubature (PS), the latter requires a pre-simulated training set. This raises a practical contradiction: if the scene has already been simulated to generate training data, there is little motivation to simulate it again with JGS2-Cubature.

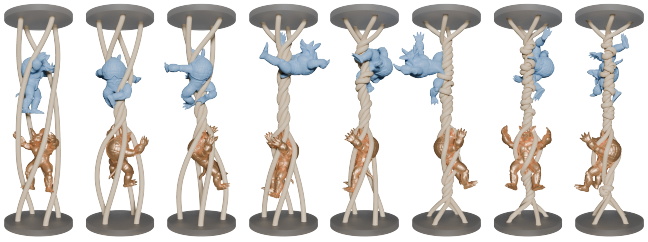


Fig. 12. **Twisted Armadillos.** Two Armadillos with distinct stiffness are twisted by four elastic rods, creating strong coupling between barrier-based friction and large nonlinear deformations. While VBD, SSD, and Chebyshev solvers fail to complete the simulation, our method maintains stability. The proposed subspace justification strategy plays an essential role for the improved performance. Compared with vanilla JGS2 [Lan et al. 2025], our method effectively reduces the average iteration count and doubles the simulation efficiency. The convergence curves are reported in Fig. 13.

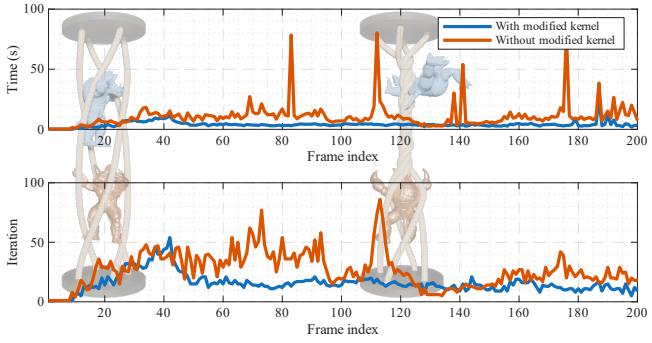


Fig. 13. **Adaptive subspace modification.** We plot the iteration count and simulation time needed for each time step with and without subspace justification for the experiment of Fig. 12. Adaptively altering subspace reduces the average iteration count from 26.7 to 15.7 and delivers a 2× speedup (10.8s to 4s per frame).

8.3 Integration vs. convergence

Using the same simulation of the Asian dragon model in Fig. 5, we quantitatively evaluate the relationship between GQ integration

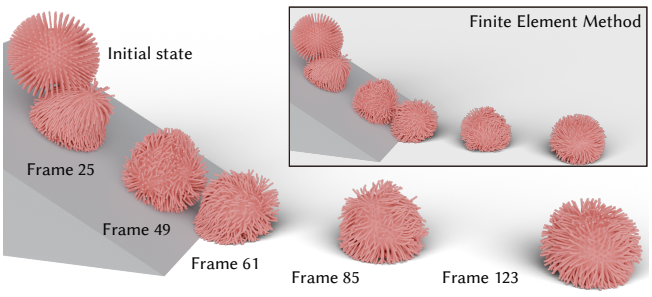


Fig. 14. **Comparison with FEM.** We compare our method against a standard FEM simulation. Our approach achieves a 33× speedup (125ms per step) while maintaining high visual fidelity and similar deformation characteristics across various frames.

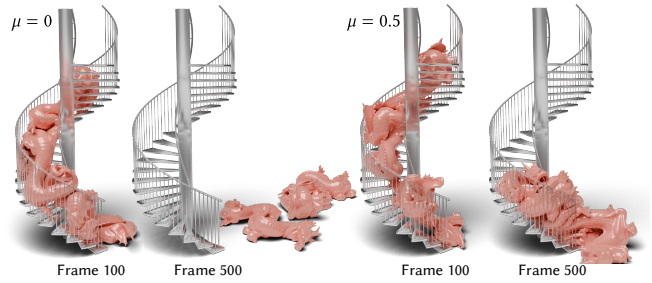


Fig. 15. **Sliding dragons.** Three dragon models slide down on a spiral staircase with $\mu = 0$ (no friction) and $\mu = 0.5$. Our method captures the energy dissipation induced by the frictional collisions, resulting in a stable pile at the base of the stairs in the frictional case. The adaptive perturbation subspace ensures that the complex contact constraints remain well-conditioned throughout the simulation.

accuracy and simulation convergence. Increasing the number of GQ points improves integration accuracy, driving the convergence up at the cost of higher runtime computation. This trend is illustrated in Fig. 6, where the number of iterations decreases as more GQ points are used. In this example, exceeding 98 GQ points does not yield further meaningful improvement in convergence. Interestingly, when considering the rate of error reduction per unit of time, a lower number of GQ points may be more efficient. Nevertheless, GQ-based JGS2 *never diverges because of unseen deformations*, and it is training-free. We would like to remind that Figs. 6 and 5 use a simulation with extreme deformations. In practice, the difference in using varying numbers of GQ points is not obvious. Using 36 or 48 GQ points is often a good default choice in practice.

8.4 Comparison with FEM

We also conducted a comparative experiment against a traditional FEM using IPC for contact handling. As illustrated in Fig. 14, which depicts a puffer ball sliding down a slope, both methods produce visually and physically consistent results. While both solvers are GPU-accelerated, our method only requires 125ms per step on average, achieving a significant 33× speedup over the FEM baseline. This performance gain is primarily attributed to our use of 2nd-order Jacobi iterations with Gaussian Quadrature. Unlike the costly global linear system solves often required in traditional FEM, our approach leverages a localized integration scheme that is highly amenable to parallelization. This allows for significantly higher throughput on the GPU while maintaining the convergence characteristics necessary for complex geometries like the puffer ball.

8.5 Comparison with other GPU algorithms

Next, we compare our method with other state-of-the-art GPU-based simulation algorithms, including Vertex Block Descent (VBD) [Chen et al. 2024b], second-order Stencil Descent (SSD) [Lan et al. 2023], and Chebyshev Jacobi [Wang and Yang 2016] without using IPC barriers. Those methods share a similar underlying parallelization modality that optimizes subproblems defined either on a node (VBD, Chebyshev) or on an element (SSD). Intuitively, those methods relax local strain within the subproblem in parallel at each iteration. The updated local information is shared via the interface of a subproblem to the neighbor DOFs at the next iteration, and a local relaxation follows. Such local information exchange is inefficient on a high-resolution model since the local solve always overshoots. Our method solves local problem bearing $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ in mind. Therefore, it is not surprising to see that our method substantially outperforms existing methods. The simulation tested is quite straightforward, where the Asian dragon deforms under gravity with its back pinned. Our convergence curve resembles Newton's method. Through GPU parallelism, JGS2-GQ delivers a clear performance advantage.

When the simulation employs IPC-based collision resolution, the nonlinearity induced by IPC barriers poses fundamental challenges for block descent-like methods. In these cases, JGS2-GQ outperforms such peers by a much larger margin, and block descent solvers are no longer our primary competitors. Instead, GPU methods tailored specifically for IPC are more relevant. To this end, we report a complex simulation as shown in Fig. 10 to illustrate the difference between our method, GPU-IPC [Guo et al. 2024], and PD-IPC [Lan et al. 2022b]. In this experiment, 140 brush models fall into a bowl. The hairs on the brush are with different stiffnesses ($E = 10^7$Pa and $E = 10^5$Pa) and interact under extensive collisions/contacts. The handle of the brush is wooden with the Young's modulus of $E = 10^9$Pa.

Existing GPU-IPC algorithms normally employ Hessian-free solve at each Newton step, such as PCG (preconditioned conjugate gradient). The key computation of this category of method is the sparse matrix-vector multiplication, which is naturally compatible with GPU parallelization. However, the bottleneck of New-PCG is the model-wise vector inner product i.e., the high-resolution discrete integration. This is exactly what our method is optimized for. Therefore, our method has a clear advantage over those methods. In the example shown in Fig. 10, our method runs 5× faster than existing GPU-IPC and over 100× faster than PD-IPC, and the convergence plots are reported in Fig. 11.

8.6 JGS2-GQ with frictional contacts

In addition to GQ-based sparse integration, our framework adaptively adjusts per-node perturbation subspaces to improve convergence during contact and friction. Fig. 12 highlights this advantage

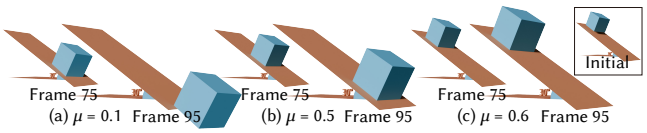


Fig. 16. **Inclined plane test.** A cube slides down from a 30° inclined plane under varying friction coefficients μ. (a) shows the initial configuration. In (b) with $\mu = 0.1$, the cube slides in an accelerated manner between frames 75 and 95. As the coefficient increases in (c) $\mu = 0.5$ and (d) $\mu = 0.6$, the sliding motion becomes more restrained accordingly. The test confirms the model's accuracy: when μ exceeds $\tan 30°$, the cube's kinetic energy is dissipated, and it comes to a complete stop, consistent with the transition from kinetic to static friction on the incline.

by comparing iteration counts with and without the subspace modification discussed in Sec. 7. In this experiment, a soft (blue) and a stiff (golden) Armadillo are tightly twisted by four elastic rods. Although the total DOF count (236K) is moderate, the scene is numerically challenging due to the strong coupling among barrier-based friction, contact, and large nonlinear deformations. While VBD, SSD, and Chebyshev solvers fail, our method maintains stability throughout the simulation. In this example, subspace justification achieves a 2× speedup, reducing average per-frame computation time from 10.8s to 4s and decreasing the average iteration count from 26.7 to 15.7. The comparative convergence plots are reported in Fig. 13

Fig. 15 provides an additional example where three dragon models slide down a spiral staircase under varying friction coefficients. When $\mu = 0$, the dragons glide unimpeded to the floor. As the coefficient is increased to $\mu = 0.5$, the kinetic energy of the falling models is significantly dissipated, causing them to decelerate and eventually stack at the base of the stairs. Another quantitative comparison is reported in Fig. 16 as a standard inclined plane test. Frictional contacts are handled with much higher accuracy and stability when the perturbation subspace is modified according to the Coulomb friction model. This adjustment ensures that local subproblems anticipate the stick-slip transitions and frictional constraints, leading to better convergence compared to the unmodified baseline.

8.7 More results

We report more simulation results using JGS2-GQ here. Most experiments involve high-resolution geometries, large variations of material stiffness, and frequent collision events. As a result, we use IPC together with CCD-based line search filtering at each JGS2 iteration.

Fig. 18 is a straightforward test, where we place Armadillo models into the bowl. Our method maintains consistent convergence under different material stiffnesses and exhibits minor fluctuations in iteration counts even under a large time step i.e., $\Delta t = 1/30$s.

Another example is shown in Fig. 17. In this experiment, we constrain the feet of eight Armadillo models to the ground and brush over them with a high-resolution, dense-bristle brush (with 500 bristles). The resulting frictional contacts and collisions cause the Armadillos to sway and collide with one another. This scenario involves intricate contact between the fine bristles and the large-scale geometries, creating complex, high-frequency impact signals. JGS2-GQ robustly manages these interactions, maintaining stable convergence even as the dense hair-to-surface collisions trigger rapid local deformations. There are 1.7M DOFs in the experiment, and our method uses 118 ms to simulate one time step.

Our method is also effective for cloth or co-dimensional simulation. When the geometry of the model degenerates, the integration domain reduces to 2D and becomes more regular. This contributes to a smaller number of quadrature points compared to volumetric models. We further evaluate this feature using a multi-layer cloth clamping experiment as shown in Fig. 19. In this setup, three square tablecloths are draped over a hollow cylinder with four deep, interior cylindrical slots. Four rigid pillars then move downwards to press the cloth layers into these slots, subjecting the material to concentrated local stretching and compression. This scenario poses a significant challenge for parallel solvers due to the dense self-collisions between the cloth layers and the high-stiffness coupling at the sharp boundaries of the slots. Despite the large membrane strains and the intricate contact configurations, JGS2-GQ resolves the deformation field robustly. The localized nature of the deformation is effectively captured by our adaptive integration domains so that the sharp stress gradients near the slot edges do not compromise global convergence.

Fig. 20 shows another example of two-way coupling between volumetric bodies and thin shells. In this experiment, nine Armadillo models are initially pulled into a high-stiffness circular elastic shell. Upon release, the shell acts as a slingshot, propelling the models at high velocity toward another square shell on the right. The Armadillos eventually fall into a chain net composed of interlocking stiff rings. This scenario incorporates diverse elements, including highly stiff thin shells, high-resolution deformable bodies, and a near-rigid chain net. Our method robustly completes the simulation despite the numerical challenges brought by large stiffness ratios and high-frequency impacts. In contrast, VBD, SSD, and Chebyshev solvers fail to maintain stability and cannot complete the simulation.

Fig. 21 demonstrates the capability of JGS2-GQ in a yarn-level simulation i.e., the tessellation of the mesh is at the yarn level. A knitted iPhone pouch consists of 1.2M elements, with each element assigned a high stiffness ($E = 10^8$Pa) to reflect the nearly inextensible nature of knitwear. As the iPhone model falls into the pouch, our method captures the intricate interactions between the cell phone and the fabric, as well as the dense self-collisions within the knit structure. JGS2-GQ robustly resolves the intense contact forces and maintains stable convergence, delivering vivid yarn-level effects that accurately represent the physical behavior of high-stiffness textiles.

Finally, we showcase our method's performance in a large-scale, high-fidelity scenario as shown in the teaser figure (Fig. 1). Ten puffer balls, each consisting of 3.8M elements, are dropped into a pumpkin-shaped container. To test numerical robustness, the orange balls are assigned a stiffness 20× greater than the green ones, creating a highly heterogeneous system. As the balls impact the interior and squeeze through the narrow opening, the elastic bristles generate massive, dense self-collisions and contact constraints. Even when integrated with the IPC framework, which is known for its high computational cost in contact-rich scenes, JGS2-GQ resolves the entire system of 5.4M DOFs with exceptional efficiency. Our method

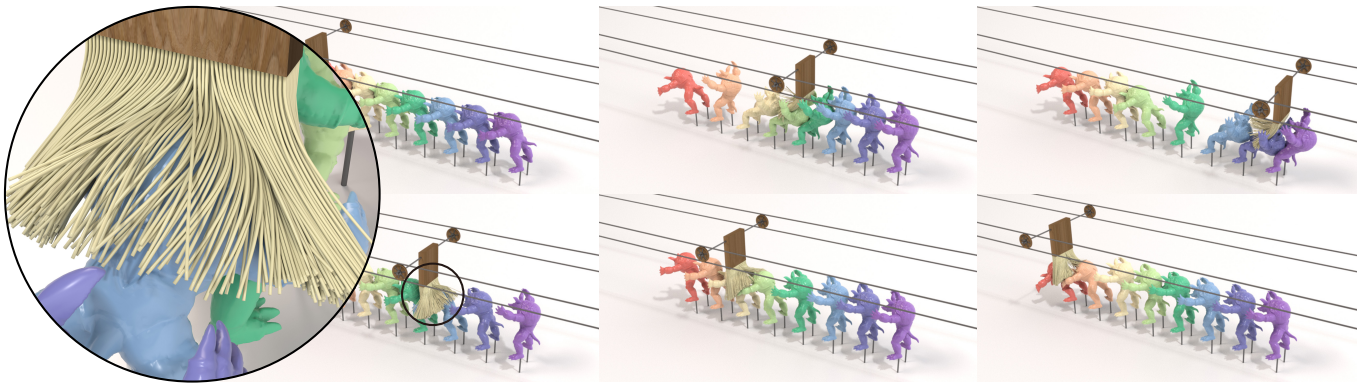


Fig. 17. **Brushing Armadillo.** Eight Armadillos are fixed on the ground at their feet, subjected to sweeping motions from a brush with 500 bristles. JGS2-GQ robustly resolves the high-frequency frictional contacts and inter-body collisions as the models sway and hit each another. There are 1.7M DOFs in the test, and our method uses on average 118 ms to simulate one frame.

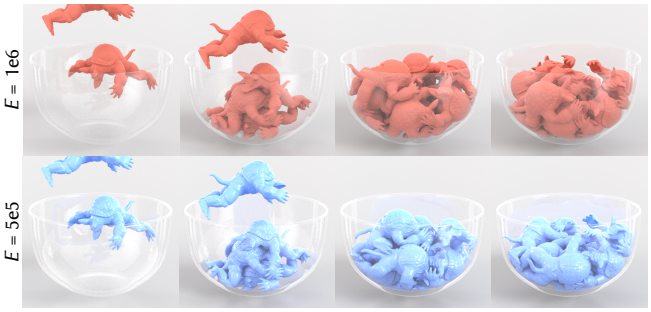


Fig. 18. **Armadillos in the bowl.** Our method is insensitive to the material stiffness and time step size. In this example, we drop 8 Armadillos into the bowl. The red Armadillo has Young's modulus of $E = 10^6$Pa while the blue one has Young's modulus of $E = 5 \times 10^5$Pa. There are 721K DOFs in this test, and our method uses equable number of iterations for each time step $\Delta t = 1/30$s.

Fig. 19. **Multilayer cloths.** Three layers of square tablecloths are draped over a cylinder and subsequently clamped into four deep slots by rigid pillars. This scenario triggers intense local stretching and complex self-collision among the layers. JGS2-GQ successfully resolves the high-stiffness coupling between the cloth and the sharp slot boundaries, maintaining stable convergence despite the large membrane strains and intricate contact configurations.

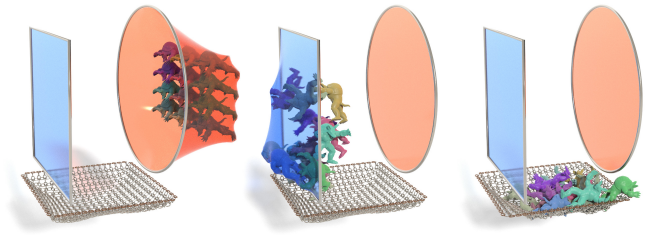


Fig. 20. **Slingshot.** This figure shows the results of a multi-stage dynamical simulation with big stiffness disparity and fast-moving impacts. Nine Armadillos are propelled by a high-stiffness "slingshot" shell and hit another elastic shell on the right before being caught by a nearly rigid-link net. JGS2-GQ robustly handles the interactions between these heterogeneous materials, whereas other peer methods (VBD, SSD, Chebyshev) are unable to complete this simulation.

maintains a stable convergence rate, requiring only 742 ms per time step, effectively bridging the gap between high-fidelity physics and high-performance GPU simulation.

8.8 Other implementation detail

Our implementation is matrix-free, and the per-node 3×3 Newton is solved analytically. The overall computational pipeline of JGS2-GQ is similar to the Jacobi method. Each node i maintains its cuboid set $\mathbf{C}_i$ along with the corresponding GQ points and weights. We restrict the integration order to a maximum of three. Most cuboids employ a one- or two-point rule according to the policy in Sec. 6.2, while $C_0(i)$ may use the three-point rule. In practice, we cap the total number of GQ points for each subproblem at 64 to prevent GPU load imbalance.

Unlike other GPU-based iterative algorithms, JGS2-GQ is insensitive to material stiffness and performs equally well for both soft and stiff materials. These properties are not a coincidence. For soft materials, the perturbation of $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{u}_i}$ is smooth and highly localized,

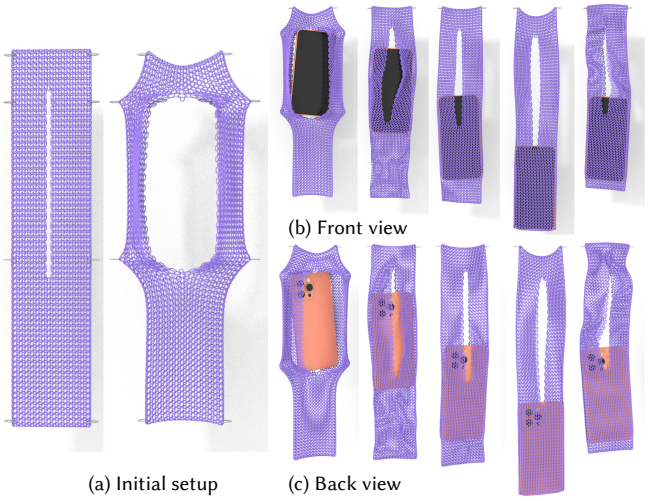


Fig. 21. **Knitwear simulation.** We show a yarn-level simulation by tessellating the tetrahedron mesh at the yarn threads of a knitted iPhone pouch. The model is composed of 1.2M elements with high stiffness to model the inextensibility of the yarn. The knitting structure involves massive self-collisions and intricate contact with the rigid device. JGS2-GQ achieves stable, high-fidelity convergence, effectively capturing the characteristic mechanical response of dense knitwear without any numerical instabilities.

allowing the GQ integration at $C_0(i)$ to accurately capture dominant global information for better convergence. Conversely, stiff materials also benefit JGS2-GQ because the polynomiality of $\frac{\partial \phi_i}{\partial u_i}$ improves, making lower-degree GQ integration more effective. This feature stands as a noteworthy advantage of JGS2-GQ over other GPU-based parallel solvers.

Dirichlet boundary conditions are common in simulation, often used to fix specific nodes on a mesh. A standard approach is to remove the corresponding rows and columns from the system matrix, and JGS2-GQ accordingly skips local solves for these constrained nodes. However, since these nodes bear strong constraint forces to maintain prescribed positions, we still build $C_0(i)$ for them and include these cuboids in the cuboid sets of other subproblems.

In general, block-descent-like methods such as PD [Bouaziz et al. 2014], XPBD [Macklin et al. 2016], and VBD [Chen et al. 2024b] prefer small time steps. JGS2-GQ is less sensitive to step size because each subproblem is globally aware, yet its convergence can still be substantially improved by a sub-stepping strategy [Macklin et al. 2019]. A conservative step adds mass-damping to $\frac{\partial \phi_i}{\partial u_i}$, which increases its polynomiality effectively. Furthermore, a smaller step suggests the perturbation is more localized, which enhances the accuracy of the integration at $C_0(i)$. This makes sub-stepping JGS2-GQ double-rewarding.

9 Conclusion & Limitation

This paper presents JGS2-GQ, a training-free framework for parallel GPU physical simulation. By replacing data-driven Cubature with an analytical integration scheme based on 3D GQ, we eliminate the requirements for offline data generation and pre-computation. We represent subproblem perturbation fields as discontinuous piecewise polynomials, allowing for the direct evaluation of reduced gradients and Hessians. This approach ensures numerical robustness even when the simulation involves novel deformations that deviate from typical training sets. Furthermore, we introduce a two-stage coupling algorithm to handle frictional contacts by dynamically adjusting the perturbation subspace. This method accounts for both primitive-level collisions and body-level elastic propagation, maintaining stability under complex contact constraints.

JGS2-GQ also has some limitations. Our current implementation is designed for meshes with a fixed topology. Because the adaptive integration regions are constructed based on the initial subproblem partitioning, the framework does not support simulations involving topological changes such as fracturing or cutting, which would necessitate re-initialization of the integration structures. Furthermore, the integration regions are currently restricted to cuboidal boxes. While efficient, these axis-aligned shapes may not optimally conform to the curved or irregular boundaries of diverse simulation models. Extending the framework to support alternative integration volumes, such as spheres or cylinders, would provide better geometric alignment with complex organic or mechanical structures.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. Yin Yang is 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. Chenfanfu Jiang is supported by NSF 2153851, TRI, Sony, and Nvidia.

References

Tomas Akenine-Möller. 2005. Fast 3D triangle-box overlap testing. In *Acm siggraph 2005 courses*. 8–es.

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 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)*. Association for Computing Machinery, New York, NY, USA, 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

Miklos Bergou, Max Wardetzky, David Harmon, Denis Zorin, and Eitan Grinspun. 2006. A quadratic bending model for inextensible surfaces. In *Symposium on Geometry Processing*, Vol. 227. 230.

Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. 2003. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph.* 22, 3 (July 2003), 917–924. doi:10.1145/882262.882364

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. Graph.* 33, 4, Article 154 (July 2014), 11 pages. doi:10.1145/2601097.2601116

Christopher Brandt, Elmar Eisemann, and Klaus Hildebrandt. 2018. Hyper-reduced projective dynamics. *ACM Trans. Graph.* 37, 4, Article 80 (July 2018), 13 pages. doi:10.1145/3197517.3201387

Claudio Canuto, M Youssuff Hussaini, Alfio Quarteroni, and Thomas A Zang. 2006. *Spectral methods: fundamentals in single domains*. Springer.

Anka He Chen, Ziheng Liu, Yin Yang, and Cem Yuksel. 2024b. Vertex Block Descent. *ACM Trans. Graph.* 43, 4, Article 116 (July 2024), 16 pages. 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.

Honglin Chen, Hsueh-Ti Derek Liu, Alec Jacobson, David I.W. Levin, and Changxi Zheng. 2024a. Trust-Region Eigenvalue Filtering for Projected Newton. In *SIGGRAPH Asia 2024 Conference Papers (SA '24)*. ACM, 1–10. doi:10.1145/3680528.3687650

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
Clark R. Dohrmann. 2003. A Preconditioner for Substructuring Based on Constrained Energy Minimization. *SIAM Journal on Scientific Computing* 25, 1 (Jan. 2003), 246–258. doi:10.1137/s1064827502412887
Charbel Farhat, Michel Lesoinne, Patrick LeTallec, Kendall Pierson, and Daniel Rixen. 2001. FETI-DP: a dual–primal unified FETI method—part I: A faster alternative to the two-level FETI method. *Internat. J. Numer. Methods Engrg.* 50, 7 (Jan. 2001), 1523–1544. doi:10.1002/nme.76
Charbel Farhat and Francois-Xavier Roux. 1991. A method of finite element tearing and interconnecting and its parallel solution algorithm. *Internat. J. Numer. Methods Engrg.* 32, 6 (Oct. 1991), 1205–1227. doi:10.1002/nme.1620320604
Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.* 35, 6, Article 214 (Dec. 2016), 9 pages. doi:10.1145/2980179.2982437
Thomas Gerstner and Michael Griebel. 2003. Dimension–adaptive tensor–product quadrature. *Computing* 71, 1 (2003), 65–87.
Dewen Guo, Minchen Li, Yin Yang, Sheng Li, and Guoping Wang. 2024. Barrier-Augmented Lagrangian for GPU-based Elastodynamic Contact. *ACM Trans. Graph.* 43, 6, Article 225 (Nov. 2024), 17 pages. doi:10.1145/3687988
Florian Hecht, Yeon Jin Lee, Jonathan R. Shewchuk, and James F. O'Brien. 2012. Updated sparse cholesky factors for corotational elastodynamics. *ACM Transactions on Graphics* 31, 5 (Aug. 2012), 1–13. doi:10.1145/2231816.2231821
P. Heyliger. 2008. Nonlinear continuum mechanics for finite element analysis (2nd edn). Javier Bonet and Richard D. Wood, Cambridge University Press, Cambridge, 2008. No. of pages: 318. ISBN: 978-0-521-83870-2. *Communications in Numerical Methods in Engineering* 24, 11 (2008), 1567–1568. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.1177 doi:10.1002/cnm.1177
Jin Huang, Lu Chen, Xinguo Liu, and Hujun Bao. 2008. Efficient mesh deformation using tetrahedron control mesh. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling (SPM08)*. ACM, 241–247. doi:10.1145/1364901.1364935
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
Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic coordinates for character articulation. In *ACM SIGGRAPH 2007 papers (SIGGRAPH07)*. ACM, 71. doi:10.1145/1275808.1276466
Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics* 24, 3 (July 2005), 561–566. doi:10.1145/1073204.1073229
Couro Kane, Jerrold E Marsden, Michael Ortiz, and Matthew West. 2000. Variational integrators and the Newmark algorithm for conservative and dissipative mechanical systems. *International Journal for numerical methods in engineering* 49, 10 (2000), 1295–1325.
Patrick Keast. 1986. Moderate-degree tetrahedral quadrature formulas. *Computer methods in applied mechanics and engineering* 55, 3 (1986), 339–348.
Lei Lan, Minchen Li, Chenfanfu Jiang, Huamin Wang, and Yin Yang. 2023. Second-order Stencil Descent for Interior-point Hyperelasticity. *ACM Trans. Graph.* 42, 4, Article 108 (July 2023), 16 pages. doi:10.1145/3592104
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 Trans. Graph.* 44, 4, Article 44 (July 2025), 15 pages. doi:10.1145/3731183
Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022a. Penetration-free projective dynamics on the GPU. *ACM Transactions on Graphics* 41, 4 (July 2022), 1–16. doi:10.1145/3528223.3530069
Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022b. Penetration-free projective dynamics on the GPU. *ACM Trans. Graph.* 41, 4, Article 69 (July 2022), 16 pages. 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 *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference 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 Trans. Graph.* 39, 4, Article 49 (Aug. 2020), 20 pages. doi:10.1145/3386569.3392425
Minchen Li, Ming Gao, Timothy Langlois, Chenfanfu Jiang, and Danny M. Kaufman. 2019. Decomposed optimization time integrator for large-step elastodynamics. *ACM Trans. Graph.* 38, 4, Article 70 (July 2019), 10 pages. doi:10.1145/3306346.3322951
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
Yaron Lipman, David Levin, and Daniel Cohen-Or. 2008. Green Coordinates. In *ACM SIGGRAPH 2008 papers (SIGGRAPH '08)*. ACM, 1–10. doi:10.1145/1399504.1360677
Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Trans. Graph.* 36, 4, Article 116a (July 2017), 16 pages. doi:10.1145/3072959.2990496
Andreas Longva, Fabian Löschner, Tassilo Kugelstadt, José Antonio Fernández-Fernández, and Jan Bender. 2020. Higher-order finite elements for embedded simulation. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–14.
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*. 49–54.
Miles Macklin, Kier Storey, Michelle Lu, Pierre Terdiman, Nuttapong Chentanez, Stefan Jeschke, and Matthias Müller. 2019. Small steps in physics simulation. In *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '19)*. Association for Computing Machinery, New York, NY, USA, Article 2, 7 pages. doi:10.1145/3309486.3340247
Jan Mandel. 1993. Balancing domain decomposition. *Communications in Numerical Methods in Engineering* 9, 3 (March 1993), 233–241. doi:10.1002/cnm.1640090307
Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-based elastic materials. In *ACM SIGGRAPH 2011 Papers* (Vancouver, British Columbia, Canada) *(SIGGRAPH '11)*. Association for Computing Machinery, New York, NY, USA, Article 72, 8 pages. 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 (2007), 109–118. doi:10.1016/j.jvcir.2007.01.005
Rahul Narain, Matthew Overby, and George E. Brown. 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
Jorge Nocedal and Stephen J Wright. 2006. *Numerical optimization*. Springer.
A. Pentland and J. Williams. 1989. Good vibrations: modal dynamics for graphics and animation. (July 1989), 215–222. doi:10.1145/74333.74355
Thomas W. Sederberg and Scott R. Parry. 1986. Free-form deformation of solid geometric models. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques (SIGGRAPH '86)*. ACM, 151–160. doi:10.1145/15922.15903
Robert W. Sumner, Johannes Schmid, and Mark Pauly. 2007. Embedded deformation for shape manipulation. In *ACM SIGGRAPH 2007 papers (SIGGRAPH07)*. ACM, 80. doi:10.1145/1275808.1276478
Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph.* 34, 6, Article 245 (Nov. 2015), 13 pages. doi:10.1145/2816795.2818081
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
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 Trans. Graph.* 34, 6, Article 246 (Nov. 2015), 9 pages. doi:10.1145/2816795.2818063
Huamin Wang and Yin Yang. 2016. Descent methods for elastic body simulation on the GPU. *ACM Trans. Graph.* 35, 6, Article 212 (Dec. 2016), 10 pages. doi:10.1145/2980179.2980236
Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A scalable galerkin multigrid method for real-time simulation of deformable objects. *ACM Trans. Graph.* 38, 6, Article 162 (Nov. 2019), 13 pages. doi:10.1145/3355089.3356486
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
Jiayi Eris Zhang, Doug James, and Danny M. Kaufman. 2024. Progressive Dynamics for Cloth and Shell Animation. *ACM Transactions on Graphics* 43, 4 (July 2024), 1–18. doi:10.1145/3658214
Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Transactions on Graphics* 29, 2 (March 2010), 1–18. doi:10.1145/1731047.1731054