JGS2: Near Second-order Converging Jacobi/Gauss-Seidel for GPU Elastodynamics

LEI LAN, University of Utah, USA
ZIXUAN LU, University of Utah, USA
CHUN YUAN, University of Utah, USA
WEIWEI XU, State Key Lab of CAD&CG, Zhejiang University, China
HAO SU, UCSD, USA
HUAMIN WANG, Style3D Research, China
CHENFANFU JIANG, UCLA, USA
YIN YANG, University of Utah, USA

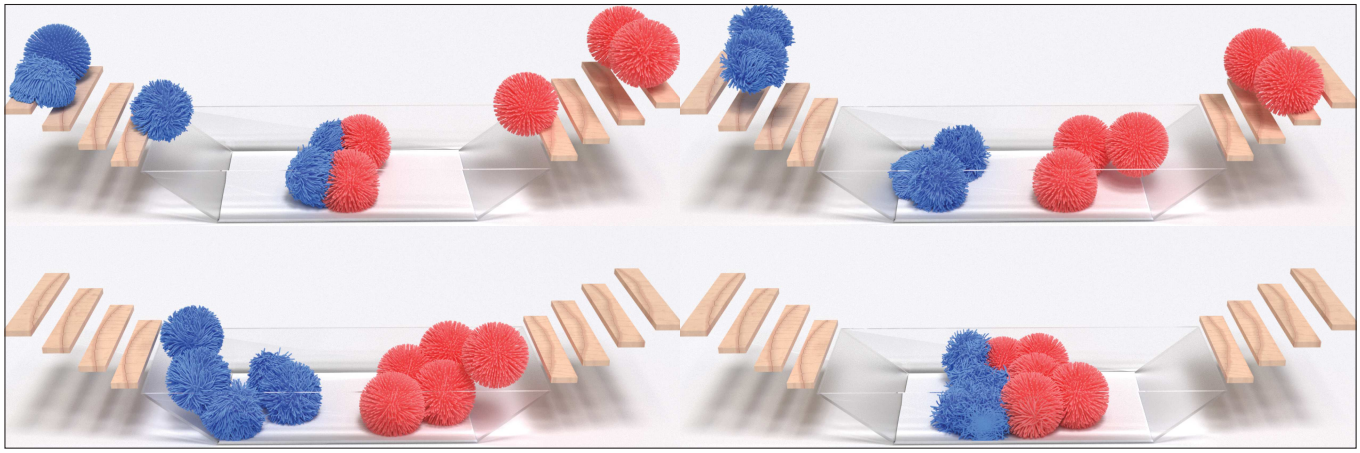
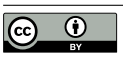


Fig. 1. **Soft and stiff puffer balls.** This paper presents a novel GPU-based parallel algorithm for elastic body simulation. We are inspired by a numerical issue of overshoot, which is the major reason behind the slow convergence of parallel solvers. Overshoot refers to the situation where local relaxation becomes over-aggressive — the reduction of local energy gets outweighed by the energy increase at other regions on the deformable object. We offer a second-order optimal solution to resolve this issue so that a parallel iteration becomes as convergent as a global Newton solve. Based on this observation, we carefully re-design the computation procedure, making this solution efficient and pre-computable. As a result, our method possesses superior parallelism (as using the Jacobi method) and near second-order convergence (as using global Newton's method). It constantly converges 50× to 100× faster than the state-of-the-art GPU methods, and our advantage is more significant for stiff simulations. The teaser figure shows a representative example. In this experiment, 10 puffer balls slide down into a glass tank. There are 3.5M tetrahedron elements in this example, and the time step size is 1/120. Blue balls are 20 times softer than red balls. Vertex block descent fails to converge at this time step. If all the puffer balls are soft ones, our method is 122× faster than vertex block descent.

In parallel simulation, convergence and parallelism are often seen as inherently conflicting objectives. Improved parallelism typically entails lighter local computation and weaker coupling, which unavoidably slow the global convergence. This paper presents a novel GPU algorithm that achieves convergence rates comparable to fullspace Newton's method while maintaining good parallelizability just like the Jacobi method. Our approach is built on a key insight into the phenomenon of *overshoot*. Overshoot occurs when a local solver aggressively minimizes its local energy without accounting for the global context, resulting in a local update that undermines global convergence. To address this, we derive a theoretically second-order optimal solution to mitigate overshoot. Furthermore, we adapt this solution into a pre-computable form. Leveraging Cubature sampling, our runtime cost is only marginally higher than the Jacobi method, yet our algorithm converges nearly quadratically as Newton's method. We also introduce a novel full-coordinate formulation for more efficient pre-computation. Our method integrates seamlessly with the incremental potential contact method and achieves second-order convergence for both stiff and soft materials. Experimental results demonstrate that our approach delivers high-quality simulations and outperforms state-of-the-art GPU methods with **50×** to **100×** better convergence.

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

Authors' Contact Information: Lei Lan, University of Utah, USA; Zixuan Lu, University of Utah, USA, birdpeople1984@gmail.com; Chun Yuan, University of Utah, USA, yuanchunisme@gmail.com; Weiwei Xu, State Key Lab of CAD&CG, Zhejiang University, China, xww@cad.zju.edu.cn; Hao Su, UCSD, USA, haosu@eng.ucsd.edu; Huamin Wang, Style3D Research, China, wanghmin@gmail.com; Chenfanfu Jiang, UCLA, USA, chenfanfu.jiang@gmail.com; Yin Yang, University of Utah, USA, yangzzzy@gmail.com.



ACM 1557-7368/2025/8-ART44
https://doi.org/10.1145/3731183

Additional Key Words and Phrases: GPU simulation, Second-order Jacobi method, Newton's method, Numerical optimization, Parallel computation

ACM Reference Format:
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 (August 2025), 15 pages. https://doi.org/10.1145/3731183

1 Introduction

Since its inception, physics-based simulation has been synonymous with high computational cost. Integrating such techniques into time-critical applications stands a significant challenge. Since then, various techniques aimed at improving simulation performance have been developed. For instance, it is possible to simplify the nonlinearity of the material to reuse a pre-factorized system matrix such as stiffness warping [Choi and Ko 2005; Müller et al. 2002], or we can build a reduced-order model using much fewer simulation degrees of freedom (DOFs) [An et al. 2008; Barbič and James 2005; Pan et al. 2015]. Despite achieving orders-of-magnitude speedups, these methods often come at the cost of compromised accuracy to some extent. In other words, they trade physical precision for performance gain. The advent of GPGPU has brought new opportunities to the field of simulation. Equipped with a large number of processing units, GPUs excel in handling massive sub computing tasks simultaneously. One-stop solvers like Newton's method using direct Hessian factorization do not fit this new computation paradigm, and nearly all the GPU algorithms opt for iterative and parallel numerical procedures. Two representative examples are the Jacobi and Gauss-Seidel (GS) methods.

Here, we consider Jacobi or GS as nonlinear relaxation schemes, where the target function is divided into small sub-problems with shared DOFs. When put into the context of quadratic optimization, they become iterative linear solvers [Greenbaum 1997]. Jacobi scheme solves each sub-problem independently. Each shared DOF has several local replicas at sub-problems, which are averaged by the end of the iteration. The classic GS routine, on the other hand, solves sub-problems sequentially — the newly updated DOF values participate in the following local solves. Parallel GS leverages graph coloring algorithms that group sub-problems without DOFs sharing so that GS updates within the group can be executed in parallel.

The key ingredient of an effective GPU simulation algorithm is always a wise trade-off between parallelization and convergence. Strategies like increasing the size of local sub-problems [Lan et al. 2023] or making sub-problems more overlapping [Luo et al. 2017] favor convergence, helping the algorithm become more effective for stiff instances. Downsizing sub-problems [Chen et al. 2024c] or delaying the information exchange [Fei et al. 2021] enhances the parallelization at the cost of more iterations or even divergence. It is believed that one can not achieve the best parallelization and convergence at the same time.

In this paper, we show a novel GPU algorithm that substantially narrows, if not closes, the gap between convergence and parallelization. Our key observation is local *overshoot* i.e., fully solving local sub-problems without knowing the global information. Overshoots negatively impact global convergence and stand as the main culprit for the slow convergence of most existing GPU algorithms. However, this issue has been overlooked and went largely unnoticed. We propose a solution that makes local computation globally aware by pre-building a reduced model for each local sub-problem. This strategy is material-aware and behaves equally well for both extremely stiff and soft problems. When the optimization problem can be well-approximated by a quadratic form, i.e., it falls within the scope of Newton's method, our approach achieves near-optimal convergence. As a result, our method converges at or near the rate of the full Newton's method while being as parallelizable as Jacobi or GS. We have tested our method in various simulation scenes. The experiment results reported are encouraging — our method converges 50× to 100× faster than the state-of-the-art GPU algorithms, paving the path to real-time and high-resolution simulation without accuracy compromises. We demonstrate the efficiency and efficacy of our algorithm in the context of elastodynamic simulation using finite element method (FEM) [Bathe 2006] however, the proposed method is readily applicable in other simulations problems such as cloth/thin shell simulation, rod simulation, MPM (material point method), and fluid simulation.

2 Related Work

High-resolution deformable bodies house a large number of two-way coupled unknown DOFs, and implicit time integration methods like backward Euler [Baraff and Witkin 1998] or Newmark [Hughes 2012] are commonly used for improved numerical stability. This results in a global (often sparse) nonlinear system. Solving this system at each time step becomes the major bottleneck of the simulation pipeline.

An effective strategy is to avoid a full linear solve in classic Newton's method. Following this idea, Hecht et al. [2012] proposed a lagged factorization scheme that reuses existing Cholesky factorization to save the computation. Chen et al. [2024b] exploited the global quadratic approximation quality to control the local eigenvalue projection in projected Newton. Multi-resolution [Capell et al. 2002b; Grinspun et al. 2002] and multigrid solvers project fine-grid residual errors onto a coarser grid, on which linear or nonlinear iterations are more effective [Bolz et al. 2003; Tamstorf et al. 2015; Wang et al. 2020; Xian et al. 2019; Zhu et al. 2010]. Quasi-Newton methods use Hessian approximates, instead of the exact Hessian, to estimate a good search direction [Li et al. 2019; Liu et al. 2017; Wang et al. 2020]. Zhang et al. [2024] took the fine-level energy into account when computing the coarse-level update in the multigrid solver for cloth and thin-shell simulation. Those methods are mostly CPU-based and seek performance gain through trading the numerical accuracy. As many graphics applications emphasize more on visual plausibility, such a trade-off is reasonable and practical.

Simplifications of the underlying elasticity model also lead to many important simulation techniques. A classic example is stiffness warping [Müller et al. 2002], which can be viewed as a simplified co-rotated material. It allows the re-use of the rest-shape stiffness matrix for rotational deformation. Stiffness warping can also be combined with modal analysis to enable real-time simulations [Choi and Ko 2005]. Chao et al. [2010] designed a simplified material model measuring the distance of linear deformation and rotation. This

concept is similar to the shape matching algorithm [Müller et al. 2005], where the deformation energy is defined based on the nearest rigid body transformation. PBD (position-based dynamics) [Müller et al. 2007] and extended PBD (XPBD) [Macklin et al. 2016] regard the elastic energy as a set of *compliant constraints* and use the steepest descent or gradient descent to update the vertex positions at each constraint. This method is later generalized for other simulation problems, including fluid [Macklin and Müller 2013], rigid bodies [Müller et al. 2020], and MPM [Yu et al. 2024]. Similarly, projective dynamics (PD) treats the elasticity energy as a collection of quadratic constraints. This assumption allows the user to separate the constraint projection and the distance measure into local and global steps [Bouaziz et al. 2014]. The key benefit of PBD and PD is the decoupling of DOFs in different constraints. As a result, both methods can be parallelized on the GPU [Fratarcangeli et al. 2016, 2018; Wang 2015]. In other words, the idealization of the underlying material model eases the solving procedure. However, the generalization of those methods to more complicated and real-world materials is less intuitive, and a careful re-formulation is needed [Macklin and Muller 2021].

Model reduction is another widely used acceleration technique, often referred to as the subspace method or reduced-order models. As the name implies, model reduction constructs a subspace representation of the fullspace DOFs. Modal analysis [Choi and Ko 2005; Hauser et al. 2003; Pentland and Williams 1989] and its first-order derivatives [Barbič and James 2005] are commonly regarded as highly effective approaches for subspace construction. Additionally, displacements from recent fullspace simulations can be leveraged to enhance the subspace representation [Kim and James 2009]. Some methods exploit condensation [Teng et al. 2015] and Schur complement [Peiret et al. 2019], which share the same nature of using a sub-set of DOFs to represent the global system status. Sheth et al. [2015] addressed the momentum conservation among contacting reduced models. The success of deep learning also brings new perspectives to simulation. Fulton et al. [2019] used an autoencoder to implicitly connect the latent space coordinate to the fullspace DOFs. Shen et al. [2021] employed complex-step finite difference (CSFD) [Luo et al. 2019] to evaluate the fictitious force caused by varying subspaces. Zong et al. [2023] built the subspace for the stress field instead of the displacement field.

A common drawback of reduced-order models lies in the lack of local details. As low-frequency deformations are normally considered more "important", and high-frequency deformations are therefore filtered by the subspace representation. This issue could be mitigated by building local subspaces. Barbič and Zhao [2011] proposed a substructuring algorithm that assumes small and nearly rigid interfaces among local subspace domains, making it particularly effective for plant simulation [Zhao and Barbič 2013]. Yang et al. [2013] integrated modal warping [Choi and Ko 2005] with component mode synthesis (CMS) [MacNeal 1971] to construct local subspaces based on interface deformations. To address locking artifacts, Kim and James [2011] introduced spring-based coupling between adjacent subspaces. Similarly, Wu et al. [2015] utilized a spring-based coupling approach, combined with Cubature [An et al. 2008] sampling, to enhance efficiency and accuracy. Harmon and Zorin [2013] augmented the subspace with local bases to capture deformation induced by collision and contact.

Previous works have also utilized coarsened geometric representations to govern the dynamics of detailed models. For example, Capell et al. [2002a] employed an embedded skeleton to deform elastic bodies, while Gilles et al. [2011] used six-DOF rigid frames to drive deformable simulations. Faure et al. [2011] introduced scattered handles to model nonlinear dynamics, and Lan et al. [2020, 2021] leveraged the medial axis transform to construct mesh skeletons. Martin et al. [2010] proposed sparsely distributed integrators, called elastons, to uniformly handle the nonlinear dynamics of rods, shells, and solids. These methods achieve significant speedups because the number of simulation DOFs is independent of the model's resolution. However, this comes at the cost of reduced accuracy and a loss of fine simulation details. After all, reduced simulation uses a low-dimension representation to model high-dimensional dynamics.

GPU simulation approaches the efficiency from a different perspective. Modern GPUs feature a large number of processors and excel in handling massive small-size computing tasks in parallel. This property requires an algorithmic re-design of simulation, shifting from a one-step solver (e.g., Newton's method) to parallelizable and iterative numerical procedures [Fratarcangeli et al. 2018]. For instance, Wang and Yang [2016] used Jacobi pre-conditioned gradient descent for elastic simulation. While the use of the Jacobi method increases the total number of iterations, the parallelized computation at the GPU compensates for it, resulting in improved overall performance. This idea can be combined with PD [Wang 2015] to solve the global step system inexactly on the GPU. Fratarcangeli et al. [2016] used a parallel GPU GS method to solve the global step matrix, which shows a better convergence than Jacobi. GS has also been a popular choice for GPU-based XPBD implementations [Chen et al. 2024a; Macklin et al. 2016].Lan et al. [2022] combined multiple Jacobi iterations into a single aggregated iteration named A-Jacobi. Guo et al. [2024] leveraged GPU for fast sparse matrix-vector computation to speed up nonlinear Newton Krylov solve. Wu et al. [2022] employed a multigrid-like pre-conditioner to further improve the convergence of GPU iterations. Similarly, subspace methods are also helpful to pre-condition the system [Lan et al. 2024; Li et al. 2023]. Despite the variety of simulation algorithm designs, GPU methods always trade convergence for parallelism, and achieving both superior convergence and parallelism is normally considered a "mission impossible". For example, to address the nonlinearity introduced by IPC (incremental potential contact) barriers [Li et al. 2020a], Lan et al. [2023] proposed a stencil descent method, which relaxes local variational energy at four vertices of an element (and a colliding primitive pair). In contrast, vertex block descent (VBD) [Chen et al. 2024c] prioritizes parallelism by solving local problems at each vertex. As a result, stencil descent performs more effectively in simulations involving stiffer objects, while VBD is better suited for softer materials.

3 Undershoot & Overshoot

We start with an explanation of overshoot and show a second-order solution to this issue. Elastodynamic simulation can be formulated as a variational optimization at each discretized time step:

$$\arg\min_{\boldsymbol{x}} E(\boldsymbol{x}). \tag{1}$$

The unknown vector $\boldsymbol{x} \in \mathbb{R}^N$ concatenates x, y, and z coordinates of all the vertices of a finite element mesh, where N denotes the system size. The target function $E = I + \Psi$ consists of the inertia potential I and the elasticity potential Ψ, which penalize accelerated motions and mesh deformation, respectively. Suppose implicit Euler integration is used, I becomes a quadratic function of $\boldsymbol{x}$: $I = \frac{1}{2h^2}\|\boldsymbol{M}^{\frac{1}{2}}(\boldsymbol{x}-\boldsymbol{z})\|^2$, where $\boldsymbol{z} = \hat{\boldsymbol{x}} + h\hat{\dot{\boldsymbol{x}}} + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_{ext}$ is a known vector depending on the previous position $\hat{\boldsymbol{x}}$, velocity $\hat{\dot{\boldsymbol{x}}}$, and an external force $\boldsymbol{f}_{ext}$. $\boldsymbol{M}$ is the mass matrix, and h is the time step size. Ψ is a nonlinear function whose specific form depends on the chosen material model and the underlying constitutive law.

We normally do not have a closed-form recipe to directly obtain $\boldsymbol{x}^\star$, the global minimizer of Eq. (1). Nearly all the nonlinear procedures start with an initial guess i.e., $\boldsymbol{x}^0$ and progressively improve this guess via $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + \delta\boldsymbol{x}^k$. Here, the superscript k denotes the iteration index. At each iteration, we would like to calculate an improving $\delta\boldsymbol{x}^k$ making $\boldsymbol{x}^{k+1}$ as close to $\boldsymbol{x}^\star$ as possible.

For GPU simulation, parallelization is commonly achieved by splitting E into multiple sub-instances or sub-problems $E_i(\boldsymbol{x}_i)$. Here, the subscript i is for the i-th sub-problem, which includes a small number of N_i DOFs, $\boldsymbol{x}_i \in \mathbb{R}^{N_i}$. An extreme case is the coordinate descent (CD) [Wright 2015], where each sub-problem contains just one unknown i.e., $N_i = 1$. When N_i is small, sub-problems can be efficiently solved in parallel using Jacobi or GS scheme. Many GPU simulation algorithms widely used in the graphics community are designed following this high-level idea e.g., see [Chen et al. 2024c; Lan et al. 2023].

A fundamental flaw of such a divide-and-conquer strategy is that *minimizing E_i does not align with minimizing E*. In other words, $\boldsymbol{x}_i^\star \neq \boldsymbol{S}_i\boldsymbol{x}^\star$, where $\boldsymbol{x}_i^\star$ is the local minimizer of E_i, and $\boldsymbol{S}_i$ is a selection matrix picking DOFs pertaining to the i-th sub-problem from the global vector. It may be possible that $\delta\boldsymbol{x}_i$ fails to sufficiently lower E_i, and therefore becomes less helpful reducing E. We refer to this issue *undershoot*. Undershoot can be potentially alleviated with a *local* line search with Wolfe condition [Wolfe 1969]. Conversely, fully relaxing E_i is also problematic because the reduction of E_i often, if not always, fails to offset the energy increases when $\delta\boldsymbol{x}_i$ is applied. In other words, locally optimal $\delta\boldsymbol{x}_i$ can worsen the overall target function E, leading to the so-called *overshoot*. Overshoot can only be monitored with global line search, which is expensive and should not be frequently used. Overshoot suggests the local computation is inaccurate since it only uses local information.

4 Towards Second-order Convergence

The ideal situation free of overshoot (and undershoot) occurs when the local solve update yields $\delta\boldsymbol{x}_i^{k+1} = \boldsymbol{S}_i\delta\boldsymbol{x}^\star$. This means that the system converges with one single iteration. Yet it is unlikely because $\delta\boldsymbol{x}^\star$ is unknown, and it is next to impossible to steer the local solve towards an uncharted target. We take a step back and aim to push the local solve to achieve global second-order convergence, i.e., at a similar rate to Newton's method.

Newton's method is well-known, which Taylor expands $E(\boldsymbol{x}^\star)$ at $\boldsymbol{x}^k$ such that:

$$\begin{aligned} E(\boldsymbol{x}^\star) = E^\star &= E(\boldsymbol{x}^k + \delta\boldsymbol{x}^k) \\ &= E^k + \boldsymbol{g}^{k\top}\delta\boldsymbol{x}^k + \frac{1}{2}\delta\boldsymbol{x}^{k\top}\boldsymbol{H}^k\delta\boldsymbol{x}^k + O(\|\delta\boldsymbol{x}^k\|^3), \end{aligned} \tag{2}$$

where $E^k = E(\boldsymbol{x}^k)$. $\boldsymbol{g}^k = \left(\frac{\partial E^k}{\partial \boldsymbol{x}}\right)^\top \in \mathbb{R}^N$, and $\boldsymbol{H}^k = \frac{\partial^2 E^k}{\partial \boldsymbol{x}^2} \in \mathbb{R}^{N\times N}$ are the gradient and Hessian of the variational energy E. If $O(\|\delta\boldsymbol{x}^k\|^3)$ is sufficiently small, and E is secondary differentiable, Newton's method converges quadratically without needing the line search [Nocedal and Wright 1999]. The corresponding DOF update $\delta\boldsymbol{x}^*$ can be computed from:

$$\delta\boldsymbol{x}^* = \arg\min_{\boldsymbol{y}} E(\boldsymbol{y}) = E^k + \boldsymbol{g}^{k\top}\boldsymbol{y} + \frac{1}{2}\boldsymbol{y}^\top\boldsymbol{H}^k\boldsymbol{y}$$

via solving the linear system of:

$$\boldsymbol{H}^k\delta\boldsymbol{x}^* = -\boldsymbol{g}^k. \tag{3}$$

Eq. (3) gives a closed-form way to compute $\delta\boldsymbol{x}^*$ so that the update $\boldsymbol{x}^* \leftarrow \boldsymbol{x}^k + \delta\boldsymbol{x}^*$ offers the second-order optimal estimation of $\boldsymbol{x}^\star$. If we make the local solve $\delta\boldsymbol{x}_i$ approach $\boldsymbol{S}_i\delta\boldsymbol{x}^*$, a parallel iteration becomes as converging as a (global) Newton step.

Let E_{C_i} be the complement of E_i i.e., the variational energy at the remaining parts of the model such that $E_{C_i} = E - E_i$. Since overshoot stems from the lack of global information, a straightforward remedy is to change the local sub-problem to:

$$\min_{\delta\boldsymbol{x}_i} E_i(\delta\boldsymbol{x}_i) + E_{C_i}(\delta\boldsymbol{x}), \tag{4}$$

so that the local solve also takes the global energy variation into account. It should be immediately noticed that $E_{C_i}(\delta\boldsymbol{x})$ depends on $\delta\boldsymbol{x}$ while the variable to be optimized in Eq. (4) only involves local DOFs $\delta\boldsymbol{x}_i$. To make Eq. (4) meaningful, we can choose to change the unknown from $\delta\boldsymbol{x}_i$ to $\delta\boldsymbol{x}$, which essentially converts Eq. (4) back to Eq. (1) — we give up the parallelization for the convergence.

Alternatively, if we know how $\delta\boldsymbol{x}_i$ would influence the global DOF variation $\delta\boldsymbol{x}$ such that $\delta\boldsymbol{x} = \phi_i(\delta\boldsymbol{x}_i)$, Eq. (4) becomes:

$$\min_{\delta\boldsymbol{x}_i} E_i(\delta\boldsymbol{x}_i) + E_{C_i}\left[\phi_i(\delta\boldsymbol{x}_i)\right], \tag{5}$$

which can then be solved e.g., using Newton's method as:

$$\delta\boldsymbol{x}_i = -\left(\boldsymbol{H}_i + \nabla E_{C_i}^\top\frac{\partial^2\phi_i}{\partial\delta\boldsymbol{x}_i^2} + \frac{\partial\nabla E_{C_i}}{\partial\delta\boldsymbol{x}_i}^\top\frac{\partial\phi_i}{\partial\delta\boldsymbol{x}_i}\right)^{-1}\left(\boldsymbol{g}_i + \frac{\partial\phi_i}{\partial\delta\boldsymbol{x}_i}^\top\nabla E_{C_i}\right), \tag{6}$$

where $\nabla E_{C_i} = \left(\frac{\partial E_{C_i}}{\partial\delta\boldsymbol{x}}\right)^\top = \left(\frac{\partial E_{C_i}}{\partial\boldsymbol{x}}\right)^\top \in \mathbb{R}^N$, and $\frac{\partial\nabla E_{C_i}}{\partial\delta\boldsymbol{x}_i} = \frac{\partial\nabla E_{C_i}}{\partial\boldsymbol{x}_i} = \frac{\partial^2 E_{C_i}}{\partial\boldsymbol{x}\partial\boldsymbol{x}_i} \in \mathbb{R}^{N\times N_i}$. Intuitively, $\phi_i(\delta\boldsymbol{x}_i)$ describes the global deformation increment caused by the local perturbation of $\delta\boldsymbol{x}_i$.

4.1 Local Perturbation Subspace

At the current Newton linearization, we compute ϕ_i by imposing a unit perturbation at one local DOF while keeping all the other local DOFs fixed. The resulting perturbation at the rest part of the model represents the influence of the perturbed local DOF. To this end, we re-order system DOFs as $\delta\boldsymbol{x} = [\delta\boldsymbol{x}_i^\top, \delta\boldsymbol{x}_{C_i}^\top]^\top$ such that $\delta\boldsymbol{x}_{C_i} \in \mathbb{R}^{N-N_i}$

contains all the *complementary* DOFs excluding the ones in $\delta \boldsymbol{x}_i$. We can then build N_i *incremental equilibria*:

$$\begin{bmatrix} \boldsymbol{H}_{i,i} & \boldsymbol{H}_{i,C_i} \\ \boldsymbol{H}_{i,C_i}^\top & \boldsymbol{H}_{C_i,C_i} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{U}_{C_i} \end{bmatrix} = \begin{bmatrix} \delta \boldsymbol{F}_i \\ \boldsymbol{0} \end{bmatrix}. \tag{7}$$

Note that the superscript k is ignored in Eq. (7). $\boldsymbol{I}$ is an N_i by N_i identity matrix. $\delta \boldsymbol{F}_i$ are the virtual forces needed to trigger the unit perturbation at each local DOF and keep others fixed. Columns in $\boldsymbol{U}_{C_i}$ embody the corresponding virtual deformation at complementary DOFs, which can be computed by expanding the second row of Eq. (7):

$$\boldsymbol{H}_{i,C_i}^\top + \boldsymbol{H}_{C_i,C_i}\boldsymbol{U}_{C_i} = \boldsymbol{0} \Rightarrow \boldsymbol{U}_{C_i} = -\boldsymbol{H}_{C_i,C_i}^{-1}\boldsymbol{H}_{i,C_i}^\top. \tag{8}$$

Any local solve update $\delta \boldsymbol{x}_i$ can be understood as a linear combination of such per-DOF perturbations i.e., $\delta \boldsymbol{x}_i = \boldsymbol{I}\delta \boldsymbol{x}_i$. Consequently, the triggered global perturbation is also a linear combination of columns in $\boldsymbol{U}_{C_i}$. In other words, $\boldsymbol{U}_{C_i}$ forms a set of bases spanning a perturbation subspace at complementary DOFs, in which the perturbations are controlled by $\delta \boldsymbol{x}_i$. Therefore, ϕ can be obtained as:

$$\delta \boldsymbol{x} = \phi_i(\delta \boldsymbol{x}_i) = \begin{bmatrix} \delta \boldsymbol{x}_i \\ \delta \boldsymbol{x}_{C_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ -\boldsymbol{H}_{C_i,C_i}^{-1}\boldsymbol{H}_{i,C_i}^\top \end{bmatrix} \delta \boldsymbol{x}_i = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{U}_{C_i} \end{bmatrix} \delta \boldsymbol{x}_i. \tag{9}$$

We note that ϕ_i becomes linearized at the current Newton step. It remains a nonlinear function during the simulation because the Hessian $\boldsymbol{H}(\boldsymbol{x}^k)$ depends on the current deformation of the system.

4.2 Optimality of ϕ_i

We argue that Eq. (9) builds the optimal local subspace so that $\delta \boldsymbol{x}_i$ computed using Eq. (6) matches the global Newton solve $\boldsymbol{S}_i \delta \boldsymbol{x}^*$. To see this, we re-organize Eq. (3) in a similar way:

$$\begin{bmatrix} \boldsymbol{H}_{i,i} & \boldsymbol{H}_{i,C_i} \\ \boldsymbol{H}_{i,C_i}^\top & \boldsymbol{H}_{C_i,C_i} \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{x}_i^* \\ \delta \boldsymbol{x}_{C_i}^* \end{bmatrix} = \begin{bmatrix} -\boldsymbol{g}_i \\ -\boldsymbol{g}_{C_i} \end{bmatrix}. \tag{10}$$

Expanding the second line, we can have:

$$\delta \boldsymbol{x}_{C_i}^* = -\boldsymbol{H}_{C_i,C_i}^{-1}\left(\boldsymbol{g}_{C_i} + \boldsymbol{H}_{i,C_i}^\top \delta \boldsymbol{x}_i^*\right). \tag{11}$$

Substituting Eq. (11) back to the first line of Eq. (10) yields:

$$\begin{aligned} &\boldsymbol{H}_{i,i}\delta \boldsymbol{x}_i^* + \boldsymbol{H}_{i,C_i}\delta \boldsymbol{x}_{C_i}^* = -\boldsymbol{g}_i \\ \Rightarrow\ &\boldsymbol{H}_{i,i}\delta \boldsymbol{x}_i^* - \boldsymbol{H}_{i,C_i}\boldsymbol{H}_{C_i,C_i}^{-1}\left(\boldsymbol{g}_{C_i} + \boldsymbol{H}_{i,C_i}^\top \delta \boldsymbol{x}_i^*\right) = -\boldsymbol{g}_i \\ \Rightarrow\ &\left(\boldsymbol{H}_{i,i} - \boldsymbol{H}_{i,C_i}\boldsymbol{H}_{C_i,C_i}^{-1}\boldsymbol{H}_{i,C_i}^\top\right)\delta \boldsymbol{x}_i^* = \boldsymbol{H}_{i,C_i}\boldsymbol{H}_{C_i,C_i}^{-1}\boldsymbol{g}_{C_i} - \boldsymbol{g}_i \\ \Rightarrow\ &\delta \boldsymbol{x}_i^* = \left(\boldsymbol{H}_{i,i} + \boldsymbol{H}_{i,C_i}\boldsymbol{U}_{C_i}\right)^{-1}\left(\boldsymbol{H}_{i,C_i}\boldsymbol{H}_{C_i,C_i}^{-1}\boldsymbol{g}_{C_i} - \boldsymbol{g}_i\right). \end{aligned} \tag{12}$$

Meanwhile, we also have the following relations:

$$\begin{cases} \dfrac{\partial \phi_i}{\partial \delta \boldsymbol{x}_i} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{U}_{C_i} \end{bmatrix}, \quad \dfrac{\partial^2 \phi_i}{\partial \delta \boldsymbol{x}_i^2} = \boldsymbol{0}, \\ \nabla E_{C_i} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{g}_{C_i} \end{bmatrix}, \quad \dfrac{\partial \nabla E_{C_i}}{\partial \delta \boldsymbol{x}_i} = \dfrac{\partial^2 E_{C_i}}{\partial \boldsymbol{x} \partial \boldsymbol{x}_i} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{H}_{i,C_i}^\top \end{bmatrix}. \end{cases} \tag{13}$$

Note that $\boldsymbol{H}_i$ in Eq. (6) is just $\boldsymbol{H}_{i,i}$ in Eqs. (7) and (10). Putting them together with Eq. (13) back to Eq. (6), we obtain:

$$\delta \boldsymbol{x}_i = \left(\boldsymbol{H}_{i,i} + \boldsymbol{H}_{i,C_i}\boldsymbol{U}_{C_i}\right)^{-1}\left(\boldsymbol{H}_{i,C_i}\boldsymbol{H}_{C_i,C_i}^{-1}\boldsymbol{g}_{C_i} - \boldsymbol{g}_i\right).$$

This shows that the local optimization using Eqs. (6) and (9) is mathematically equivalent to solving the global Newton and satisfies $\delta \boldsymbol{x}_i = \boldsymbol{S}_i \boldsymbol{x}^*$. In other words, the resulting $\delta \boldsymbol{x}_i$ is second-order optimal.

4.3 Co-rotated Subspace

Computing ϕ_i^k via Eq. (9) needs the current Hessian matrix $\boldsymbol{H}^k(\boldsymbol{x}^k)$, which varies under different mesh poses. Clearly, it is infeasible to re-build ϕ_i^k at each time step.

Recall that ϕ_i helps avoid overshoot because it brings awareness of E_{C_i} during the local solve. That said, the key to mitigating overshoot lies in how well $E_{C_i}(\phi_i^k)$ is estimated. This requirement is less stringent than demanding $\phi_i^k = \delta \boldsymbol{x}^*$. The latter always guarantees that $E_{C_i}(\phi_i^k)$ is exact; the reverse, however, does not always hold true. Therefore, it suffices to construct alternative subspace function $\tilde{\phi}_i^k$ such that $E_{C_i}(\tilde{\phi}_i^k)$ closely matches $E_{C_i}(\phi_i^k)$ even though $\tilde{\phi}_i^k(\delta \boldsymbol{x}_i)$ may not exactly align with $\phi_i^k(\delta \boldsymbol{x}_i)$.

The idea is to embed a co-rotated local frame at each mesh vertex. Given the current deformation pose $\boldsymbol{x}^k$, we extract a local rotation $\boldsymbol{R}_{\langle j \rangle}^k \in SO(3)$ at vertex j. Here, we use the subscripted notation $\langle j \rangle$ to denote the vertex index. $\boldsymbol{R}_{\langle j \rangle}^k$ can be efficiently computed by applying the polar decomposition over the local deformation gradient at all the vertices in parallel. This local rotation captures how an infinitesimal chunk of material around the vertex j is rotated w.r.t. its rest pose. As a rigid rotation preserves the elasticity energy, we rotate the vertex back to this rest orientation without modifying the energy it stores. Meanwhile, ϕ at the rest shape can be pre-computed. Mathematically, this strategy builds $\tilde{\phi}_i^k$ as:

$$\tilde{\phi}_i^k = \boldsymbol{R}^k \underbrace{\begin{bmatrix} \boldsymbol{I} \\ -\bar{\boldsymbol{H}}_{C_i,C_i}^{-1}\bar{\boldsymbol{H}}_{i,C_i}^\top \end{bmatrix}}_{\bar{\boldsymbol{U}}_i} \boldsymbol{R}_i^{k\top} \delta \boldsymbol{x}_i^k = \underbrace{\boldsymbol{R}^k \bar{\boldsymbol{U}}_i \boldsymbol{R}_i^{k\top}}_{\boldsymbol{U}_i^k} \delta \boldsymbol{x}_i^k, \tag{14}$$

where $\boldsymbol{R}^k$ and $\boldsymbol{R}_i^k$ are two block-diagonal matrices whose 3 by 3 diagonal blocks are per-vertex local rotation matrix. At the current Newton step, $\boldsymbol{R}^k$ and $\boldsymbol{R}_i^k$ are constant. $\bar{\boldsymbol{H}}_{C_i,C_i}^{-1}\bar{\boldsymbol{H}}_{i,C_i}^\top$ is also constant depending on the rest-shape Hessian $\bar{\boldsymbol{H}}$. In other words, $\tilde{\phi}_i^k$ remains a linearized subspace at the current Newton linearization, and it can also be efficiently constructed at different deformation poses because $\bar{\boldsymbol{H}}_{C_i,C_i}^{-1}\bar{\boldsymbol{H}}_{i,C_i}^\top$ is pre-computable.

4.4 Discussion

The subspace function ϕ_i^k plays a central role. It allows global awareness by predicting how the global energy changes in response to the local update $\delta \boldsymbol{x}_i$. As a result, the local solve well aligns with $\boldsymbol{S}_i \delta \boldsymbol{x}^*$. ϕ_i^k does not reduce the size of the local problem so that the local solve remains in the N_i-rank space. However, the remainder part of the optimization (E_{C_i}) is condensed to a subspace whose generalized subspace coordinate is designed to be $\delta \boldsymbol{x}_i$. While this is a nonlinear subspace that varies w.r.t. deformation poses, it can be linearized at each local quadratic approximation of $\boldsymbol{x}^\star$. We design an alternative formulation $\tilde{\phi}_i^k$ to allow pre-computation for the most expensive part of the subspace construction. As a result, the reduced Hessian of $\frac{\partial^2 E_{C_i}}{\partial \delta \boldsymbol{x}_i^2}$ acts as a "damper" preventing the local solver from reaching its local minimizer (and thus overshoots).

The subspace function ϕ_i^k can also be viewed as a type of interpolation function that smoothly propagates $\delta\boldsymbol{x}_i$ to $\delta\boldsymbol{x}$. To this end, many interpolation algorithms may also help such as radial basis function (RBF) [Botsch and Kobbelt 2005; Carr et al. 2001], Green coordinates [Lipman et al. 2008; Michel and Thiery 2023], Splines [Li et al. 2023; Liu et al. 2014], Harmonics [Jacobson et al. 2011; Lipman et al. 2010], or SPH kernels [Koschier et al. 2022]. However, those methods are geometry-based and do not reflect the material property. For instance, $\delta\boldsymbol{x}_i$ produces more general and global perturbations for stiff materials and more local and regional perturbations for soft materials. The perturbations at x, y, or z coordinates are also different given different constitutive models. After all, geometry-based interpolation schemes are not designed to make local solve second-order optimal i.e., $\delta\boldsymbol{x}_i = \boldsymbol{S}_i\delta\boldsymbol{x}^*$. On the other hand, ϕ performs like a material-aware shape function, expanding the influence of a sub-problem towards the entire object.

5 Cubature Sampling

$\tilde{\phi}_i^k$ is linearized at the current deformation poses $\boldsymbol{x}^k$, and we have $\frac{\partial\tilde{\phi}_i^k}{\partial\delta\boldsymbol{x}_i} = \boldsymbol{U}_i^k$ and $\frac{\partial^2\tilde{\phi}_i^k}{\partial\delta\boldsymbol{x}_i^2} = \boldsymbol{0}$ per Eq. (14). Substituting them into Eq. (6) with some manipulations gives the local system we need to solve:

$$\left(\boldsymbol{H}_{i,i}^k + \tilde{\boldsymbol{H}}_{i,i}^k\right)\delta\boldsymbol{x}_i^k = -\tilde{\boldsymbol{g}}_i^k - \boldsymbol{g}_i^k, \tag{15}$$

where

$$\tilde{\boldsymbol{H}}_{i,i}^k = \boldsymbol{U}_i^{k^\top}\left(\nabla^2 E_{C_i}^k\right)\boldsymbol{U}_i^k, \quad \tilde{\boldsymbol{g}}_i^k = \boldsymbol{U}_i^{k^\top}\nabla E_{C_i}^k, \tag{16}$$

are the reduced Hessian and gradient force. Eq. (15) is of low dimension and can be efficiently solved at each GPU thread. However, its assembly is not, since $\boldsymbol{U}_i^k$ is a dense matrix. To exactly build $\tilde{\boldsymbol{H}}_{i,i}^k$ and $\tilde{\boldsymbol{g}}_i^k$, one needs to traverse all the complementary DOFs and project their Hessian and gradient into the column space of $\boldsymbol{U}_i^k$. The time complexity is $O(N\cdot N_i^2)$, and it needs to be done for all the sub-problems.

A known solution was proposed in [An et al. 2008] a.k.a. Cubature. Cubature is a sampling technique, which pre-computes a small group of sample elements $\mathcal{S}_i$ i.e., Cubature elements, and the associated non-negative weights. The reduced Hessian and gradient force are then approximated by:

$$\tilde{\boldsymbol{H}}_{i,i}^k \approx \sum_{e\in\mathcal{S}_i} w_e[\boldsymbol{U}_i^k]_e^\top[\nabla^2 E_{C_i}^k]_e[\boldsymbol{U}_i^k]_e,\ \tilde{\boldsymbol{g}}_i^k \approx \sum_{e\in\mathcal{S}_i} w_e[\boldsymbol{U}_i^k]_e^\top[\nabla E_{C_i}^k]_e, \tag{17}$$

where $[\nabla E_{C_i}^k]_e \in \mathbb{R}^{12}$ and $[\nabla^2 E_{C_i}^k]_e \in \mathbb{R}^{12\times 12}$ are the gradient and Hessian of $E_{C_i}^k$ at the Cubature element e. $[\boldsymbol{U}_i^k]_e \in \mathbb{R}^{12\times N_i}$ is the element subspace matrix, which extracts corresponding rows of e from $\boldsymbol{U}_i^k$. w_e is the non-negative weight coefficient.

Given a set of training poses $\mathcal{T}$, the corresponding reduced gradient (i.e., $\tilde{\boldsymbol{g}}_i^{(1)}$,..., $\tilde{\boldsymbol{g}}_i^{(|\mathcal{T}|)}$), and Cubature element set $\mathcal{S}_i$, the weight coefficients at all the Cubature elements are computed via solving:

$$\begin{bmatrix} \frac{[\tilde{\boldsymbol{g}}_i^{(1)}]_1}{\|\tilde{\boldsymbol{g}}_i^{(1)}\|} & \cdots & \frac{[\tilde{\boldsymbol{g}}_i^{(1)}]_{|\mathcal{S}_i|}}{\|\tilde{\boldsymbol{g}}_i^{(1)}\|} \\ \vdots & \cdots & \vdots \\ \frac{[\tilde{\boldsymbol{g}}_i^{(|\mathcal{T}|)}]_1}{\|\tilde{\boldsymbol{g}}_i^{(|\mathcal{T}|)}\|} & \cdots & \frac{[\tilde{\boldsymbol{g}}_i^{(|\mathcal{T}|)}]_{|\mathcal{S}_i|}}{\|\tilde{\boldsymbol{g}}_i^{(|\mathcal{T}|)}\|} \end{bmatrix}\begin{bmatrix} w_1 \\ \vdots \\ w_{|\mathcal{S}|_i} \end{bmatrix} = \begin{bmatrix} \frac{\tilde{\boldsymbol{g}}_i^{(1)}}{\|\tilde{\boldsymbol{g}}_i^{(1)}\|} \\ \vdots \\ \frac{\tilde{\boldsymbol{g}}_i^{(|\mathcal{T}|)}}{\|\tilde{\boldsymbol{g}}_i^{(|\mathcal{T}|)}\|} \end{bmatrix}, \tag{18}$$

using non-negative least square (NNLS). Several candidate Cubature elements are randomly picked from the non-Cubature elements, and the one most effectively reduces the residual of Eq. (18) will be included in $\mathcal{S}_i$. This procedure continues until the desired residual error is reached. We refer the reader to the seminal paper of Cubature [An et al. 2008] for further details.

Cubature training is normally considered expensive, and acceleration techniques are also available e.g., see [Von Tycowicz et al. 2013; Yang et al. 2015]. The complexity of Cubature training is largely due to repetitive NNLS solving, which scales up super-polynomially when more Cubature samples are needed. The size of Cubature sample set is linearly correlated with the size of the subspace i.e., $|\mathcal{S}_i| \propto N_i$. As a result, we only need a handful Cubature elements e.g., four or six, for each sub-problem in our implementation (with residual less than 1%), making Cubature training lightweight.

We would also like to mention that unlike most existing algorithms for reduced simulation, which aim to construct a compact subspace for *deformation* or *displacement*, our subspace depicts deformable *perturbation* $\delta\boldsymbol{x}$. Therefore, $\boldsymbol{U}_i^k$ does not need to incorporate large deformation. To this end, our training poses are simply low-frequency eigenvectors of the rest-shape Hessian matrix $\bar{\boldsymbol{H}}$. In addition, what we need is a good estimation of E_{C_i}. The exactness of Cubature gradient is of less importance – this is because our primary goal is to estimate a reasonable "damping Hessian" to prevent overshoot other than exactly minimize E. As a result, sparse Cubature sampling is highly effective in our framework.

6 Full-coordinate Pre-computation

The co-rotated formulation allows the most expensive step in constructing $\tilde{\phi}_i^k$ to be pre-computed as shown in Eq. (14). For each sub-problem i, the pre-computation solves $\bar{\boldsymbol{H}}_{C_i,C_i}$ for N_i times. $\bar{\boldsymbol{H}}_{C_i,C_i}$ is a $(N-N_i)\times(N-N_i)$ matrix. It is nearly of the same scale as $\bar{\boldsymbol{H}}$ since N_i is a small quantity. Performing such factorization for all the sub-problems is extremely slow, which takes days for large-scale models. We observe that while $\bar{\boldsymbol{H}}_{C_i,C_i}$ differs across different sub-problems, a significant portion of these matrices overlaps. This suggests a brute-force computation would be inefficient and wasteful. Motivated by this observation, we propose a full-coordinate formulation for Eq. (7), which accelerates the pre-computation by *three orders*, reducing the time required from days to tens of minutes.

We know that Eq. (7) builds $\tilde{\phi}_i^k$ via a set of incremental equilibria. Each column of $\delta\boldsymbol{F}_i$ embodies an external force at local DOFs, which triggers a unit perturbation at a specific local DOF while keeping other local DOFs fixed. The very same equilibrium can also be achieved by directly prescribing local DOF values with N_i position constraints:

$$\bar{\boldsymbol{H}}\bar{\boldsymbol{u}}_{i,j} = \boldsymbol{0}, \quad \text{s.t. } \boldsymbol{S}_i\bar{\boldsymbol{u}}_{i,j} = \boldsymbol{e}_j, \text{ for } j = 1, 2, \cdots, N_i, \tag{19}$$

where $\boldsymbol{e}_j$ is the j-th column of $\boldsymbol{I}$, which is a vector of zeros with a value of one at the j-th entry. $\bar{\boldsymbol{u}}_{i,j} \neq \boldsymbol{0}$ is the deformation variation of the mesh induced by the constraint. The subscript i, j suggests constraints are imposed for the i-th sub-problem, whose j-th local DOF is perturbed. This constrained linear system can be solved with

the full coordinate and Lagrange multipliers such that:

$$\begin{bmatrix} \bar{\boldsymbol{H}} & \boldsymbol{S}_i^\top \\ \boldsymbol{S}_i & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{U}}_i \\ \boldsymbol{\Lambda}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}. \tag{20}$$

Here, $\bar{\boldsymbol{U}}_i = [\bar{\boldsymbol{u}}_{i,1}, \cdots, \bar{\boldsymbol{u}}_{i,N_i}]$. $\boldsymbol{\Lambda}_i = -\delta \boldsymbol{F}_i$ are Lagrange multipliers. Compared with Eq. (7), Eq. (20) solves both unknown DOFs and multipliers, and it is normally considered less efficient. However, because its top-left block becomes invariant for all the sub-problems, we can leverage block-wise matrix inversion to re-use factorized $\bar{\boldsymbol{H}}$ without performing factorization of different $\bar{\boldsymbol{H}}_{C_i,C_i}$ repeatedly.

The block-inverse of the l.h.s. of Eq. (20) gives:

$$\begin{bmatrix} \bar{\boldsymbol{H}} & \boldsymbol{S}_i^\top \\ \boldsymbol{S}_i & \boldsymbol{0} \end{bmatrix}^{-1} = \begin{bmatrix} \bar{\boldsymbol{H}}^{-1} + \bar{\boldsymbol{H}}^{-1}\boldsymbol{S}_i^\top \bar{\boldsymbol{G}}_i \boldsymbol{S}_i \bar{\boldsymbol{H}}^{-1} & -\bar{\boldsymbol{H}}^{-1}\boldsymbol{S}_i^\top \bar{\boldsymbol{G}}_i \\ -\bar{\boldsymbol{G}}_i \boldsymbol{S}_i \bar{\boldsymbol{H}}^{-1} & \bar{\boldsymbol{G}}_i \end{bmatrix}, \tag{21}$$

where $\bar{\boldsymbol{G}}_i = -\left(\boldsymbol{S}_i \bar{\boldsymbol{H}}^{-1} \boldsymbol{S}_i^\top\right)^{-1}$ is the inverse of Schur complement of $\bar{\boldsymbol{H}}$. Inverting Eq. (20) leads:

$$\begin{bmatrix} \bar{\boldsymbol{U}}_i \\ \boldsymbol{\Lambda}_i \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{H}} & \boldsymbol{S}_i^\top \\ \boldsymbol{S}_i & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}, \tag{22}$$

which shows that:

$$\bar{\boldsymbol{U}}_i = -\bar{\boldsymbol{H}}^{-1} \boldsymbol{S}_i^\top \bar{\boldsymbol{G}}_i. \tag{23}$$

During the pre-computation, we only factorize $\bar{\boldsymbol{H}}$ once, which is shared by all the sub-problems without factorizing different $\bar{\boldsymbol{H}}_{C_i,C_i}$ at individual sub-problems. $\bar{\boldsymbol{G}}_i \in \mathbb{R}^{N_i \times N_i}$ is a small matrix and can be efficiently calculated by solving $\bar{\boldsymbol{H}}$ for N_i times. As a result, the pre-computation of a sub-problem only needs $2N_i$ forward and backward substitutions of $\bar{\boldsymbol{H}}$, and such computation can be trivially parallelized at all the sub-problems.

7 Incremental Potential Contact

Collisions among deformable objects can also be considered as a type of potential energy and uniformly encoded in the variational optimization. A representative paradigm is the incremental potential contact or IPC [Li et al. 2020a]. IPC is a primal implementation of the interior-point method, which injects a logarithmic barrier energy into Eq. (1), defined at each surface primitive pair l:

$$B_l(d_l, \hat{d}) = \begin{cases} -(d_l - \hat{d})^2 \log\left(\frac{d_l}{\hat{d}}\right), & 0 < d_l < \hat{d} \\ 0, & d_l \geq \hat{d} \end{cases}. \tag{24}$$

$\hat{d}$ is a user-specified parameter prescribing the collision tolerance. B_l becomes "active" if the closest distance between a collision pair (i.e., d_l) is smaller than $\hat{d}$, and it approaches to $+\infty$ as d_l approaches to 0. Intuitively, IPC offers a nonlinear penalty mechanism pushing a pair of colliding primitives, e.g., a vertex-triangle pair or an edge-edge pair, away from each other when they are in proximity.

Our method approaches improved convergence from an optimization point of view. As a result, it is compatible with IPC or other implicit penalty methods as long as the collision resolution is in the form of an unconstrained optimization. The key adaptation is to accommodate ϕ_i with collisions and contacts. When a vertex V on model A is in contact with another object B ϕ_i does not only concern the total energy on A but also has a non-vanishing influence on the energy on B. In other words, A and B become two-way coupled by the contact at the vertex. The value of the subspace basis on A is pre-computed. While the values of ϕ_i at B can be approximated by assuming all the colliding vertices on B have the same perturbation as V. This strategy assumes IPC or the contact penalty is much stiffer than the elasticity stiffness, and the variation of the perturbation within the sub-problem is ignored.

8 Experimental Results

We implemented our pipeline on a desktop computer with an intel i7-12700 CPU (for pre-computation) and an Nvidia 3090 RTX GPU. We used Spectra library for computing the eigendecomposition of the rest-shape Hessian matrix $\bar{\boldsymbol{H}}$. It should be noted that our framework can be conveniently deployed with other parallel computing platforms, such as multi-core CPUs. Nevertheless, this section reports the simulation performance and results based on our GPU implementation. Tab. 1 lists detailed experiment setups and timing information. For all the examples, we normalize the scene into a one-by-one-by-one unit cube and use the change of position of the object between two consecutive iterations i.e., $\|\Delta \boldsymbol{x}\|$ as a simple and uniformed measure for convergence check. Please also refer to the supplementary video for more animation results.

8.1 Parallel Implementation

Our method is generic and does not impose restrictions on how a sub-problem should be specified. In our implementation, we assign a sub-problem at each mesh vertex, making $N_i = 3$. The local Newton relaxation needs to tackle a 3-by-3 system, which can be analytically computed. In theory, the positive definiteness of the local Newton system should be taken care of as explained in [Smith et al. 2018]. We note that the local solve is regularized by $\tilde{\boldsymbol{H}}_{i,i}$ i.e., see Eq. (15), which is sampled at multiple remote Cubature elements. In practice, we would not worry about numerical issues of our local solve since this reduced Hessian is always well-conditioned.

The implementation of Jacobi parallelization is straightforward. When sub-problems are at vertices, one Jacobi iteration solves all the sub-problems at vertices in parallel. No averaging is needed. However, if the sub-problem is defined for multiple vertices e.g., at an element as in [Lan et al. 2023], a Jacobi iteration also averages duplicated DOFs shared by neighboring sub-problems. Because our local solve is nearly optimal $\delta \boldsymbol{x}_i \approx \boldsymbol{S}_i \delta \boldsymbol{x}^*$, bigger-size sub-problems do not improve the convergence obviously, and lighter local solve should be favored. GS parallelization puts independent sub-problems into groups using graph coloring algorithms [Fratarcangeli et al. 2016]. One GS iteration traverses all the sub-problems of all groups. If GS parallelization is used, we consider all the sub-problems within one group as a *generalized sub-problem* and pre-compute the corresponding $\tilde{\phi}_i^k$ for each group as a whole. Because all the sub-problems within a group are independent, the local Hessian is block-diagonal. In other words, the pre-computation effort is as light as the Jacobi parallelization. The only difference lies in the computation of $\bar{\boldsymbol{U}}_i$. When GS parallelization is chosen, the constraint in Eq. (19) applies one unit perturbation at a specific DOF of the current generalized sub-problem, while keeping all the other DOFs of the generalized sub-problem fixed. Nevertheless, we do not observe any noticeable difference between Jacobi and GS parallelization — both converge at the rate of Newton's method even under large time steps.

Table 1. **Experiment statistics.** This table reports time statistics and simulation setups for all the experiments mentioned in the paper. # **Element** gives the total number of elements in the example. # **DOF** is the total number of simulation DOFs. $|\mathcal{S}_i|$ is the size of Cubature samples used for each sub-problem, knowing that each sub-problem has three DOFs. $\|\Delta \boldsymbol{x}\|$ is the convergence condition. h gives the time step size used for the simulation. **Parallelism** tells if the parallelization is in a Jacobi way or a GS way i.e., Jacobi or GS. The difference between those two parallelization methods is negligible. The column of **Collision** shows what method is used for collision resolution, using either the implicit penalty method (Penalty) or incremental potential contact (IPC). # **Iteration** reports the average number of iterations needed to simulate a time step. **Pre. time** is the total time used for pre-computation, and **Sim. time** is the average simulation time for simulating one time step. We also report the acceleration rate our method offers compared with VBD [Chen et al. 2024c] if the collision is processed with the penalty method or GPU-IPC [Guo et al. 2024] if the collision is processed with the IPC barrier. We use stable Neo-Hookean model [Smith et al. 2018] for all the deformable body experiments. For cloth simulation result i.e., Fig. 13, we use StVK model to capture the in-plane deformation, and quadratic bending for the out-of-plane deformation.

Scene	# Element	# DOF	$\lvert\mathcal{S}_i\rvert$	h	$\|\Delta \boldsymbol{x}\|$	Parallelism	Collision	# Iteration	Pre. time	Sim. time
Teaser (Fig. 1)	3.5M	4.4M	4	1/120	$1E-3$	GS	Penalty	55	37 min.	855 ms (∞×)
Falling Armadillos (Fig. 4)	6M	3.6M	4	1/150	$5E-4$	Jacobi	Penalty	58	52 min.	883 ms (78×)
House of cards (Fig. 5)	394K	372K	4	1/50	$1E-3$	Jacobi	IPC	23	1 min.	31 ms (120×)
Dragon (Fig. 6)	100K	80K	6	1/100	$1E-3$	Jacobi	Penalty	9	7 min.	7.3 ms (32×)
Letters soft (Fig. 7)	2.1M	1.7M	6	1/120	$5E-4$	GS	Penalty	27	37 min.	176 ms (43×)
Barbarian ships (Fig. 8)	2.5M	2.1M	4	1/120	$3E-4$	Jacobi	Penalty	34	45 min.	333 ms (153×)
Jack-o′-lanterns (Fig. 9)	6.7M	5.7M	4	1/120	$5E-4$	GS	Penalty	32	4 min.	753 ms (40×)
Squeezed puffer ball (Fig. 10)	1.3M	0.9M	6	1/150	$3E-4$	Jacobi	Penalty	69	67 min.	290 ms (173×)
Cactus (Fig. 11)	1.2M	1M	4	1/150	$1E-3$	Jacobi	IPC	40	18 min.	171 ms (82×)
Animal corossing (Fig. 12)	4.8M	4.5M	4	1/150	$1E-3$	GS	IPC	43	10 min.	684 ms (136×)
Cloth (Fig. 13)	2M	3M	4	1/120	$1E-3$	Jacobi	IPC	42	48 min.	469 ms (103×)

8.2 Overshoot Comparison

We illustrate the issue of overshoot and compare our method with several well-known parallelable algorithms, including XPBD [Macklin et al. 2016], projective dynamics (PD) [Bouaziz et al. 2014], vertex block descent (VBD) [Chen et al. 2024c], and second-order stencil descent (2nd SD) [Lan et al. 2023]. XPBD and PD are widely known for their efficiency and convenient parallelization. A limitation of XPBD or PD is their reliance on the idealization of the material. To make the comparison objective, we use the as-rigid-as-possible (ARAP) material [Igarashi et al. 2005], which can be naturally handled with XPBD and PD.

The direct way to quantify overshoot is to measure the difference between $\boldsymbol{x}_i$ and $\boldsymbol{S}_i\boldsymbol{x}^*$ for sub-problem i. We use a standard simulation setup, where a rectangular beam with one end fixed at the wall bends down under its gravity. At a specific frame, we simulate the mesh displacement for the next time step with $h = 1/100$ using global Newton's method. The global convergence condition is set as the relative residual force error being smaller than $1E-4$. The resulting deformation $\boldsymbol{x}^*$ serves as the reference for this comparison. After that, we use different algorithms to simulate the same frame with the same material parameters. We pick one tetrahedron element and plot the variation of $\|\boldsymbol{x}_i - \boldsymbol{S}_i\boldsymbol{x}^*\|$ w.r.t. the number of parallel iterations using different methods.

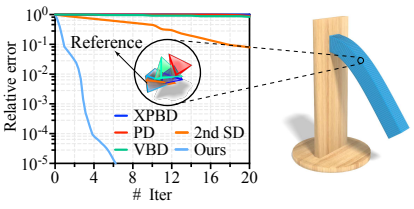


Fig. 2. **Overshoot of local solvers.** We plot the relative error between $\boldsymbol{x}_i$ and $\boldsymbol{S}_i\boldsymbol{x}^*$ such that $\boldsymbol{x}_i$ is the position of a tetrahedron element obtained by different local solvers, including XPBD [Macklin et al. 2016], PD [Bouaziz et al. 2014], VBD [Chen et al. 2024c], 2nd SD [Lan et al. 2023], and our method. Our method converges as fast as Newton's method does, and the error reaches $1E-3$ with just three iterations. The other methods barely make progress even after 20 iterations.

The curves highlight the overshoot problem of existing methods as shown in Fig. 2. Our method shows a strong quadratic convergence and pushes $\boldsymbol{x}_i$ to the reference with only a handful of iterations. On the other hand, existing methods such as XPBD, PD, and VBD barely improve $\boldsymbol{x}_i$ even after 20 iterations. 2nd SD, due to its bigger sub-problem size and hybrid parallelization scheme, delivers a better convergence. But it still gets outperformed by our method by a significant margin. In fact, it will take over 500 VBD, PD, or XPBD iterations in this example to reduce the relative error to the order of $1E-3$, whereas our method only needs three iterations. The cost of our method for completing one iteration is slightly higher than VBD or XPBD because of extra computations for Cubature elements. Overall, our method is more than 50× faster than XPBD, PD, VBD or 2nd SD in this example.

8.3 Comparison with VBD & 2nd SD

We regard VBD from Chen et al. [2024c] and 2nd SD from Lan et al. [2023] as our most relevant competitors. Both VBD and 2nd SD use Newton's method to handle sub-problems. The key difference is that VBD sets a sub-problem as a vertex (i.e., same as our implementation), while 2nd SD uses a tetrahedron element as a sub-problem. They offer different trade-offs: VBD has better parallelization but converges slower for large time steps or stiff materials. 2nd SD converges better, but local solve is much more expensive, which solves a 12 by 12 system. As shown previously in Fig. 2, both VBD and 2nd SD suffer from the overshoot problem.

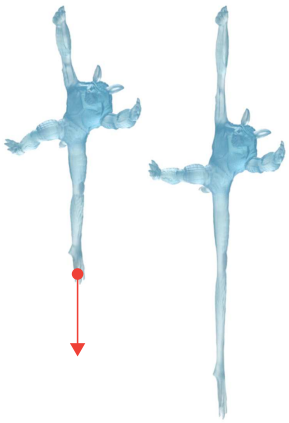


Fig. 3. **Comparison with VBD & 2nd SD.** The Armadillo model consists of 1M vertices and 3.4M elements. We fix its left hand and drag the right leg downwards to produce a large-scale body deformation. We record the total number of iterations needed for this example using VBD [Chen et al. 2024c], 2nd SD [Lan et al. 2022], and our method under different material stiffness with $h = 1/100$. Our method is 15× faster than 2nd SD and 40× faster than VBD. After further increasing the stiffness of the Armadillo by 20 times, our method is 34× faster than 2nd SD, and 137× faster than VBD (VBD switches to a highly conservative time step of $h = 1/20000$).

To further elaborate on the difference among those peer parallelization strategies, we simulate an Armadillo model with its left hand fixed by applying a sharp and big force at the right foot to generate large-scale body deformation (Fig. 3). There are 1M vertices and 3.4M elements on the model. We compare the average number of iterations needed to simulate one time step using VBD, 2nd SD, and our method under different material stiffness with $h = 1/100$. The convergence condition is set as $\|\Delta \boldsymbol{x}\| < 1E-4$. The material model is stable Neo-Hookean [Smith et al. 2018]. On average, it takes 34 Newton iterations to simulate one time step, and our method needs 38 (Jacobi) iterations. The Newton method takes approximately 150 seconds per iteration to perform the Cholesky decomposition using the MKL PARDISO solver, while our method takes only 11 ms to complete an iteration. 2nd SD needs 74 hybrid iterations. 2nd SD is about 15× more expensive than our method at each iteration. Therefore, our method is $\sim$ 30× faster than 2nd SD. VBD needs 2, 264 iterations to converge one time step, and our method is 40× faster. We then increase the stiffness of the Armadillo for 20 times and run the same simulation with these methods keeping $h = 1/100$. In this case, the Newton iteration count increases to 58, and our method needs 64 iterations. 2nd SD uses 142 iterations on average, and VBD fails to converge even after 10, 000 iterations. VBD becomes convergent when h is reduced to $1/20000$, and it still needs more than 440 iterations for one time step. This is equivalent to using 8, 800 iterations to simulate $1/100$ real-world seconds. In this example, our method is 137× faster. The performance of VBD becomes even worse under intensive collisions.

Fig. 4. **Falling Armadillos.** Six Armadillo models consist of 1.2M vertices and 6M elements. They collide with each other while falling into a container. Our method requires an average of 58 iterations per time step with $h = 1/150$, compared to 44 iterations for the projected Newton method. However, our method requires only 883 ms per step, achieving a speedup of approximately 8000× compared to the projected Newton method, which takes 106 minutes per step.

It now becomes clear that the excellent performance of VBD heavily relied on small time steps and soft materials. 2nd SD offers a more robust solution for stiff simulations. Unfortunately, both VBD and 2nd SD suffer from overshoot. While our method is orders-of-magnitude faster. The advantage becomes more noticeable in stiffer simulations.

8.4 Comparison with Projected Newton Method

We also compare our method with the projected Newton method using GPU-based direct solvers. In each iteration, the Newton method performs Cholesky decomposition of a large-scale sparse linear system, which leads to significant computational overhead. In contrast, our method decomposes the global system into smaller 3 by 3 subsystems, achieving significant acceleration by fully exploiting GPU parallelism.

As shown in Fig. 4, we simulate six Armadillo models falling into a container, where Armadillos undergo mutual collisions. These six Armadillos contain 1.2M vertices and 6M elements. At this scale, a Cholesky decomposition used for the Newton method takes approximately 150 seconds. We use $h = 1/150$ and resolve collisions using the implicit penalty method. The projected Newton method requires an average of 44 iterations to complete a simulation step. Our method requires 58 iterations under the Jacobi parallelization

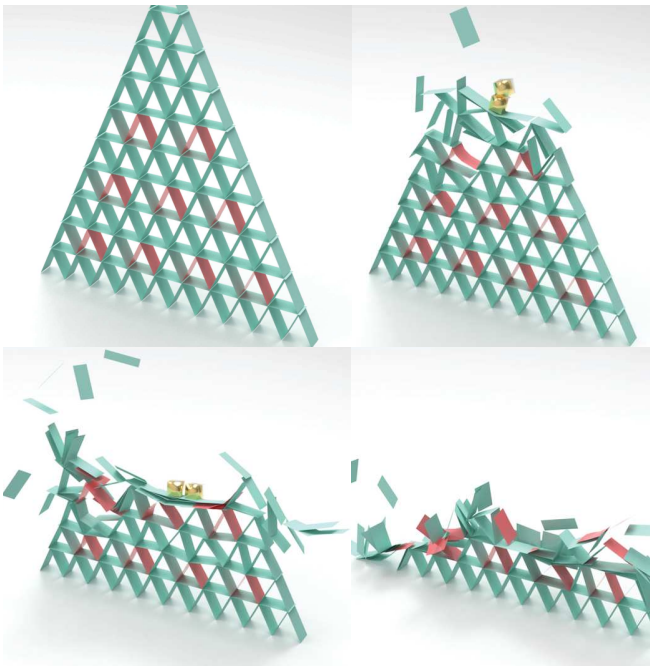


Fig. 5. **House of cards.** A stack of 155 cards is initially balanced through frictional contacts using IPC barriers [Li et al. 2020a]. The house of cards collapses under a high-velocity impact from two boxes. Each card has 2,543 elements. The greed cards are 200 times more stiffer than red ones. Our method takes 31 ms to simulate one frame, which is over 1,000× faster than CPU-based Newton IPC. VBD does not converge in this example.

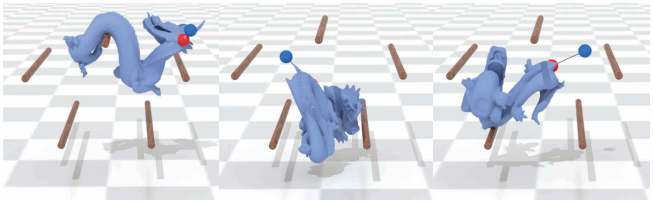


Fig. 6. **Real-time simulation.** Our method enables real-time simulation of complex deformable bodies. The dragon has 100K elements, and it can be simulated in real time under user manipulations. With $h = 1/100$, our method takes fewer than 10 iterations to simulate one frame. The runtime simulation exceeds 120 FPS in this example, including collision detection.

(e.g., need slightly more iterations compared with Newton). However, each iteration only needs 15 ms. As a result, our method is approximately 8,000× faster than the projected Newton method.

8.5 Collision

Our method is compatible with existing collision processing algorithms such as IPC [Li et al. 2020a] or penalty method [Wu et al. 2020], as long as the simulation can be formulated as an unconstrained optimization. An example is shown in Fig. 5, where a "house of card" is structured by 155 cards. The green cards are 200 times more stiffer than red cards. They stack each other the frictional contacts. Two heavy cubes fall, and the card stacking collapses by this external impact. There are 394K elements in this simulation. Our method uses 31 ms to simulate one time step ($h = 1/50$), which is over 1,000× faster than CPU-based IPC simulation and more than 120× faster than Newton-Krylov-based GPU solvers [Guo et al. 2024]. VBD does not converge in this problem due to the existence of a highly nonlinear barrier (even under $h = 1/5000$).

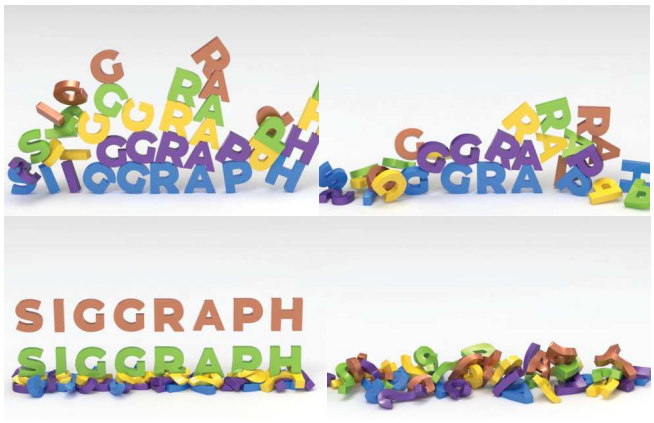


Fig. 7. **Stiff and soft letters.** Five sets of "SIGGRAPH" letters fall on the floor. The letters on the top row are 1,000 times more stiffer than the ones in the bottom row. Our method is not sensitive to the variation of material stiffness. For each time step, it needs 34 iterations for the stiff letters and 27 iterations for the softer letters. 2nd SD and VBD fail to converge when simulating the stiff letters.

8.6 Soft & Stiff Simulations

Our method exploits $\tilde{\phi}$ to estimate the global energy variation during the local solve. Solving $\bar{\boldsymbol{H}}_{C_i,C_i}$ incorporates the material properties of the body at complementary DOFs. This makes our method less sensitive to material variations. Meanwhile, most known parallel algorithms prefer softer simulations over stiffer simulations because DOFs are more strongly coupled with stiff materials, and the local optimum of a sub-problem is more likely to overshoot (e.g., see the comparison of Fig. 3). In addition to Fig. 5, we show two more examples in Figs. 1 and 7 involving both soft and stiff objects. In Fig. 1, we simulate the dynamics of ten puffer balls sliding from stairs into a glass container from both sides. There are 3.5M elements in this example. Blue puffer balls are 20 times softer than the red puffer balls. Our method needs 55 iterations to simulate one time step ($h = 1/120$). VBD does not converge in this example even under $h = 1/360$. If all the puffer balls are soft ones, VBD becomes converging but is 122× slower than our method.

Another example is reported in Fig. 7, where we drop five sets of "SIGGRAPH" letters on the ground. The letters on the top are 1,000 times stiffer than the ones at the bottom. Our method handles both simulations stably using similar numbers of iterations i.e., 27 iterations for soft letters and 34 iterations for hard letters). Meanwhile, neither VBD nor 2nd SD converges for hard letters.

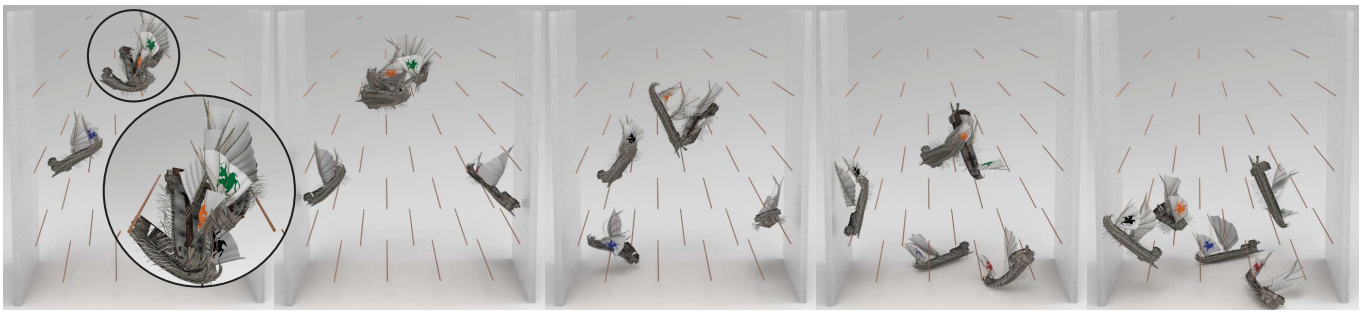


Fig. 8. **Barbarian ships.** Five barbarian ships fall and interact with multiple thin rods between two walls. There are more than 2.5M elements in this example. Our method uses 333 ms to simulate one time step using penalty forces. Our simulation is 153× faster than VBD.

Fig. 9. **Jack-o′-lanterns.** 850 Halloween jack-o′-lanterns fall into a container with a deformable tree, and they fully bury the tree eventually. There are 6.7M elements in this example in total. Our method takes 753 ms for each time frame, which is 40× faster than VBD.

8.7 Real-time Simulation

Our method enables real-time elastic simulation of complicated shapes of real-world material. Fig. 6 shows snapshots of screen records of a real-time simulation of a dragon. The user interactively manipulates the Neo-Hookean dragon [Smith et al. 2018], which consists of 100K elements, and the simulation runs in real-time at more than 100 FPS following the user inputs.

8.8 More Results

We show more large-scale simulation results using our method. All of these examples involve complex shapes and high-resolution models. Fig. 8 reports an example of simulating five barbarian ships. Each barbarian ship has 500K elements, and the simulation handles 2.5M elements. Under $h = 1/120$, our method needs 34 iterations on average, and the solving time for each time step is 333 ms. This is 153× faster than VBD. In Fig. 9, we keep dropping deformable Jack-o′-lanterns into a container with an old tree until the tree is fully buried by the lanterns. There are 6.7M DOFs and 850 pumpkins in this simulation. Our method is 40× faster than VBD.

The reader may notice the different performance gains in those two examples, and the improvement of our method tends to become less pronounced in Fig. 9. This is because deformable bodies in Fig. 9 do not have a large number of elements compared with the ship model in Fig. 8. Isolated DOFs on different objects are naturally decoupled, and the local solver is less likely to overshoot. To verify this, we show another challenging simulation in Fig. 10. In this example, we simulate a puffer ball with 1.2M elements (i.e., it is of higher resolution than the puffer ball model used in Fig. 1) falling into an elastic net of 588 rings and 329K elements. The net twists and squeezes the puffer ball. In this example, our method is 173× faster than VBD.

Our method works with IPC as well [Li et al. 2020a]. An example is given in Fig.11, where six bone dragons fall into a cactus bush with high velocities. The impacts from the dragon trigger significant dynamics at the cactus. The use of IPC ensures the simulation is free of interpenetration, and frictional contacts between the bone dragons and cactus are also accurately captured. Compared with vanilla IPC, which solves the global Newton at each iteration, our method is about $5,000\times$ faster. This speedup is a rough estimation, as we have not completed this simulation on CPU IPC. Compared with GPU-IPC [Guo et al. 2024], our method is 82× faster. There are 1M DOFs in the simulation, and our method takes 171 ms to simulate one frame. Another example is shown in Fig. 12, where 500 small animals drop into a tank. After that, we use a glassy plane to press all the animals and release the constraint suddenly, making all the animal toys bounce back up. Those little toys are of different stiffness. In this example, there are 4.8M elements, and it is 70× faster than GPU projective dynamics [Lan et al. 2022].

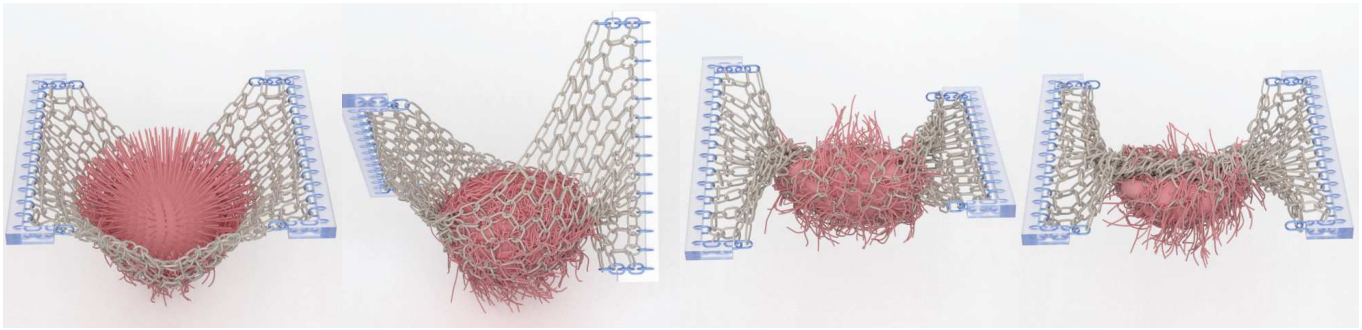


Fig. 10. **Squeeze puffer ball.** In this simulation, we drop a puffer ball into a soft elastic chain. The puffer ball, comprising 1.2M elements, interacts with an elastic net made up of 588 rings and 329K elements, connected via ring-ring contacts. As the ball descends, the net twists and compresses it, demonstrating the effects of tightly coupled contacts and elasticity. This simulation runs with a time step of $h = 1/150$, and each time step takes 290 ms. Our method is 173× faster than VBD.

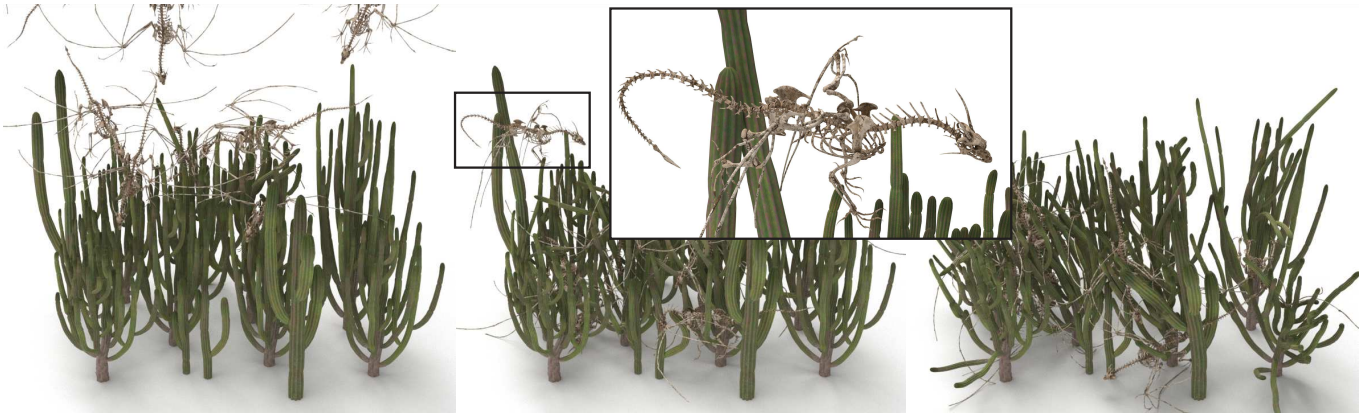


Fig. 11. **Bone dragons from the sky.** Six bone dragons fall into the cactus bush. There are 1.2M elements in the simulation, and we use IPC to robustly process high-velocity collisions between bone dragons and cacti, as well as self-collisions among cacti. In this example, our method is over 82× faster than GPU-based IPC simulation [Guo et al. 2024].

Our method is also able to handle co-dimensional models like thin-shell and cloth. To this end, we show the result of a high-resolution cloth simulation in Fig. 13. The tablecloth consists of 2M triangles and over 3M DOFs. It is displaced to cover a helicopter model from the top. Our method captures detailed wrinkles of the cloth during the movement, and it converges with about 42 iterations on average. In this example, our method is three orders faster than CPU-based C-IPC [Li et al. 2020b].

9 Conclusion & Future Work

In this paper, we explore the problem of overshoot in parallel deformable simulation. When overshoot occurs, the local solver over-aggressively reduces its local target, and the resulting DOFs' update negatively impacts the target function at other areas on the deformable body. We give a second-order optimal solution to avoid overshoot and make this procedure pre-computable. This leads to a new GPU simulation algorithm that possesses both excellent parallelism and (near) second-order convergence. We have tested our method on a wide range of large-scale simulation scenes. Our method constantly outperforms existing GPU simulation algorithms by orders, making real-time simulation of complicated deformable objects possible.

Our method also has some limitations. Our method needs to build a local subspace at each sub-problem and carry out Cubature training for co-rotationed basis vectors. Such pre-computation is slow and could impose practical inconvenience. The second-order convergence depends on the quadratic approximation of the global optimal $E^\star$ i.e., see Eq. (2). When Newton's approximation is less appropriate and $\|\delta \boldsymbol{x}^k\|^3$ is a relatively big quantity, our method does not converge quadratically. This could happen when the variational optimization involves highly nonlinear terms such as the IPC barrier. Line search is then needed. Fortunately, line search can be done at each sub-problem in parallel. As a result, our method remains orders-of-magnitude faster than global IPC solvers. While our algorithm pushes the simulation performance to a new level, collision detection becomes the new bottleneck. We demonstrate the feasibility and potential of our method in the context of hyperelastic simulation, yet the proposed algorithm is actually a general-purpose parallel

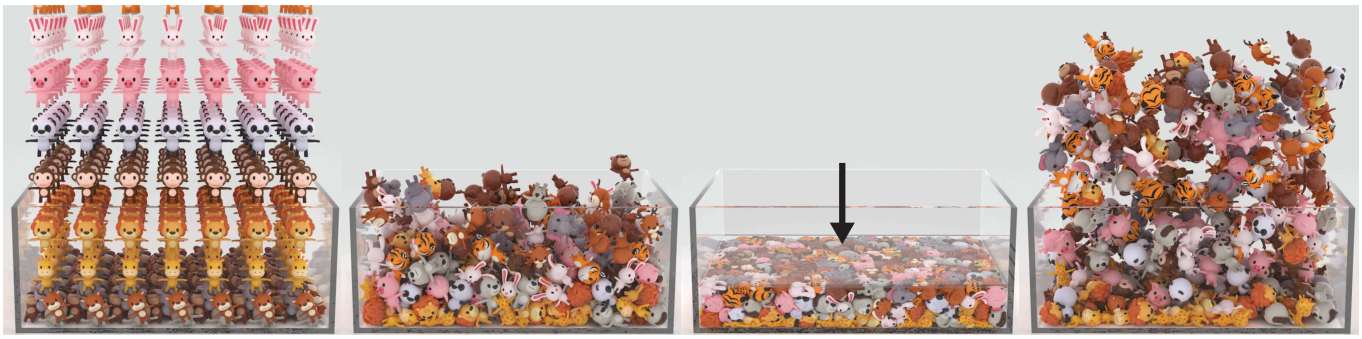


Fig. 12. **Animal crossing.** 500 small animal toys fall into the tank, and there are 4.8 elements in this example. A glass plane is then pushed down to compress all these little toys. After the removal of the plane, the compressed animals bounce back into the air. Animals have different stiffness Our method takes 684 ms to simulate one time step, which is 70× faster than GPU projective dynamics [Lan et al. 2022], and 136× faster than GPU-IPC [Guo et al. 2024].

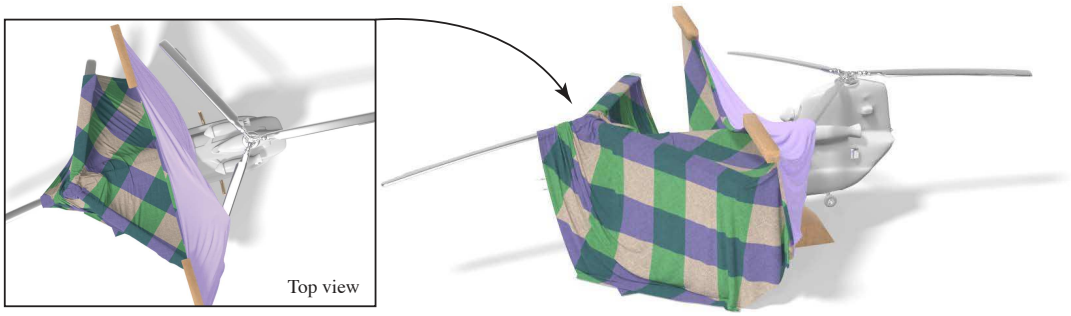


Fig. 13. **Cover the helicopter.** Our method is not limited to deformable simulation and can also be readily used for thin-shell and cloth simulation. In this example, a piece of tablecloth covers a helicopter. It is displaced back and forth, generating detailed wrinkles. There are 3M DOFs in the simulation. We use IPC to process collisions between the cloth and the helicopter as well as the self-collision on the cloth. It takes 469 ms to simulate one frame, and our method is 12, 00× faster than co-dimensional IPC [Li et al. 2020b].

optimization procedure. Therefore, it is of great interest for us to apply our method in other simulation and graphics problems. The key difficulty is still how to find a pre-computable ϕ to efficiently and effectively estimate the global energy variation. We believe a data-driven approach, i.e., a deep learning perspective may be a good answer to this challenge, which could offer a case-by-case optimized setup for different computational problems.

Acknowledgments

We thank reviewers for their detailed and constructive comments. Chenfanfu Jiang is partially supported by NSF 2153851 and TRI. Yin Yang is partially supported by NSF under grant number 2301040.

References

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

David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 43–54.

Jernej Barbič and Doug L James. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM transactions on graphics (TOG)* 24, 3 (2005), 982–990.

Jernej Barbič and Yili Zhao. 2011. Real-time large-deformation substructuring. *ACM transactions on graphics (TOG)* 30, 4 (2011), 1–8.

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

Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. 2003. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM transactions on graphics (TOG)* 22, 3 (2003), 917–924.

Mario Botsch and Leif Kobbelt. 2005. Real-time shape editing using radial basis functions. In *Computer graphics forum*, Vol. 24. Blackwell Publishing, Inc Oxford, UK and Boston, USA, 611–621.

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

Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. 2002a. Interactive skeleton-driven dynamic deformations. In *ACM Trans. Graph. (TOG)*, Vol. 21. ACM, 586–593.

Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. 2002b. A multiresolution framework for dynamic deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 41–47.

Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. 2001. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 67–76.

Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. 2010. A simple geometric model for elastic deformations. *ACM transactions on graphics (TOG)* 29, 4 (2010), 1–6.

Anka He Chen, Ziheng Liu, Yin Yang, and Cem Yuksel. 2024c. Vertex Block Descent. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–16.

Honglin Chen, Hsueh-Ti Derek Liu, Alec Jacobson, David IW Levin, and Changxi Zheng. 2024b. Trust-Region Eigenvalue Filtering for Projected Newton. In *SIGGRAPH Asia 2024 Conference Papers*. 1–10.

Yizhou Chen, Yushan Han, Jingyu Chen, Zhan Zhang, Alex Mcadams, and Joseph Teran. 2024a. Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–15.

Min Gyu Choi and Hyeong-Seok Ko. 2005. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics* 11, 1 (2005), 91–101.

François Faure, Benjamin Gilles, Guillaume Bousquet, and Dinesh K Pai. 2011. Sparse meshless models of complex deformable solids. In *ACM Trans. Graph. (TOG)*, Vol. 30. ACM, 73.

Yun Fei, Yuhan Huang, and Ming Gao. 2021. Principles towards real-time simulation of material point method on modern GPUs. *arXiv preprint arXiv:2111.00699* (2021).

Marco Fratarcangeli, Valentina Tibaldo, Fabio Pellacini, et al. 2016. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.* 35, 6 (2016), 214–1.

Marco Fratarcangeli, Huamin Wang, and Yin Yang. 2018. Parallel iterative solvers for real-time elastic deformations. In *SIGGRAPH Asia 2018 Courses*. 1–45.

Lawson Fulton, Vismay Modi, David Duvenaud, David IW Levin, and Alec Jacobson. 2019. Latent-space Dynamics for Reduced Deformable Simulation. In *Computer graphics forum*, Vol. 38. Wiley Online Library, 379–391.

Benjamin Gilles, Guillaume Bousquet, Francois Faure, and Dinesh K Pai. 2011. Frame-based elastic models. *ACM Trans. Graph. (TOG)* 30, 2 (2011), 15.

Anne Greenbaum. 1997. *Iterative methods for solving linear systems*. SIAM.

Eitan Grinspun, Petr Krysl, and Peter Schröder. 2002. CHARMS: A simple framework for adaptive simulation. *ACM transactions on graphics (TOG)* 21, 3 (2002), 281–290.

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

David Harmon and Denis Zorin. 2013. Subspace integration with local deformations. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

Kris K Hauser, Chen Shen, and James F O'Brien. 2003. Interactive Deformation Using Modal Analysis with Constraints.. In *Graphics Interface*, Vol. 3. 16–17.

Florian Hecht, Yeon Jin Lee, Jonathan R Shewchuk, and James F O'Brien. 2012. Updated sparse cholesky factors for corotational elastodynamics. *ACM Trans. Graph. (TOG)* 31, 5 (2012), 123.

Thomas JR Hughes. 2012. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation.

Takeo Igarashi, Tomer Moscovich, and John F Hughes. 2005. As-rigid-as-possible shape manipulation. *ACM transactions on Graphics (TOG)* 24, 3 (2005), 1134–1141.

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4 (2011), 78.

Theodore Kim and Doug L James. 2009. Skipping steps in deformable simulation with online model reduction. In *ACM Trans. Graph. (TOG)*, Vol. 28. ACM, 123.

Theodore Kim and Doug L James. 2011. Physics-based character skinning using multi-domain subspace deformations. In *Proceedings of the 2011 ACM SIGGRAPH/eurographics symposium on computer animation*. 63–72.

Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. 2022. A survey on SPH methods in computer graphics. In *Computer graphics forum*, Vol. 41. Wiley Online Library, 737–760.

Lei Lan, Minchen Li, Chenfanfu Jiang, Huamin Wang, and Yin Yang. 2023. Second-order Stencil Descent for Interior-point Hyperelasticity. *ACM Transactions on Graphics* 42, 4 (2023).

Lei Lan, Zixuan Lu, Jingyi Long, Chun Yuan, Xuan Li, Xiaowei He, Huamin Wang, Chenfanfu Jiang, and Yin Yang. 2024. Efficient GPU Cloth Simulation with Non-distance Barriers and Subspace Reuse. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–16.

Lei Lan, Ran Luo, Marco Fratarcangeli, Weiwei Xu, Huamin Wang, Xiaohu Guo, Junfeng Yao, and Yin Yang. 2020. Medial Elastics: Efficient and Collision-Ready Deformation via Medial Axis Transform. *ACM Transactions on Graphics (TOG)* 39, 3 (2020), 1–17.

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

Lei Lan, Yin Yang, Danny Kaufman, Junfeng Yao, Minchen Li, and Chenfanfu Jiang. 2021. Medial IPC: accelerated incremental potential contact with medial elastics. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–16.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy R Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. 2020a. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.* 39, 4 (2020), 49.

Minchen Li, Ming Gao, Timothy Langlois, Chenfanfu Jiang, and Danny M. Kaufman. 2019. Decomposed Optimization Time Integrator for Large-Step Elastodynamics. *ACM Transactions on Graphics* 38, 4 (2019).

Minchen Li, Danny M Kaufman, and Chenfanfu Jiang. 2020b. Codimensional incremental potential contact. *arXiv preprint arXiv:2012.04457* (2020).

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*. 1–12.

Yaron Lipman, David Levin, and Daniel Cohen-Or. 2008. Green coordinates. *ACM transactions on graphics (TOG)* 27, 3 (2008), 1–10.

Yaron Lipman, Raif M Rustamov, and Thomas A Funkhouser. 2010. Biharmonic distance. *ACM Transactions on Graphics (TOG)* 29, 3 (2010), 1–11.

Songrun Liu, Alec Jacobson, and Yotam Gingold. 2014. Skinning cubic Bézier splines and Catmull-Clark subdivision surfaces. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 1–9.

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

Ran Luo, Weiwei Xu, Tianjia Shao, Hongyi Xu, and Yin Yang. 2019. Accelerated complex-step finite difference for expedient deformable simulation. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–16.

Ran Luo, Weiwei Xu, Huamin Wang, Kun Zhou, and Yin Yang. 2017. Physics-based quadratic deformation using elastic weighting. *IEEE transactions on visualization and computer graphics* 24, 12 (2017), 3188–3199.

Miles Macklin and Matthias Müller. 2013. Position based fluids. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

Miles Macklin and Matthias Muller. 2021. A constraint-based formulation of stable neo-hookean materials. In *Proceedings of the 14th ACM SIGGRAPH conference on motion, interaction and games*. 1–7.

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.

Richard H MacNeal. 1971. A hybrid method of component mode synthesis. *Computers & Structures* 1, 4 (1971), 581–601.

Sebastian Martin, Peter Kaufmann, Mario Botsch, Eitan Grinspun, and Markus Gross. 2010. Unified simulation of elastic rods, shells, and solids. In *ACM Trans. Graph. (TOG)*, Vol. 29. ACM, 39.

Élie Michel and Jean-Marc Thiery. 2023. Polynomial 2D Green coordinates for polygonal cages. In *ACM SIGGRAPH 2023 Conference Proceedings*. 1–9.

Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. 2002. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 49–54.

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.

Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. 2005. Meshless deformations based on shape matching. In *ACM Trans. Graph. (TOG)*, Vol. 24. ACM, 471–478.

Matthias Müller, Miles Macklin, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim. 2020. Detailed rigid body simulation with extended position based dynamics. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 101–112.

Jorge Nocedal and Stephen J Wright. 1999. *Numerical optimization*. Springer.

Zherong Pan, Hujun Bao, and Jin Huang. 2015. Subspace dynamic simulation using rotation-strain coordinates. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–12.

Albert Peiret, Sheldon Andrews, József Kövecses, Paul G Kry, and Marek Teichmann. 2019. Schur complement-based substructuring of stiff multibody systems with contact. *ACM Transactions on Graphics (TOG)* 38, 5 (2019), 1–17.

Alex Pentland and John Williams. 1989. Good vibrations: Modal dynamics for graphics and animation. In *SIGGRAPH Comput. Graph.*, Vol. 23. ACM.

Siyuan Shen, Yin Yang, Tianjia Shao, He Wang, Chenfanfu Jiang, Lei Lan, and Kun Zhou. 2021. High-order differentiable autoencoder for nonlinear model reduction. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–15.

Rahul Sheth, Wenlong Lu, Yue Yu, and Ronald Fedkiw. 2015. Fully momentum-conserving reduced deformable bodies with collision, contact, articulation, and skinning. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 45–54.

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

Rasmus Tamstorf, Toby Jones, and Stephen F McCormick. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph. (TOG)* 34, 6 (2015), 245.

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

Christoph Von Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. 2013. An efficient construction of reduced deformable objects. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–10.

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

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

Xinlei Wang, Minchen Li, Yu Fang, Xinxin Zhang, Ming Gao, Min Tang, Danny M Kaufman, and Chenfanfu Jiang. 2020. Hierarchical optimization time integration for cfl-rate mpm stepping. *ACM Transactions on Graphics (TOG)* 39, 3 (2020), 1–16.

Philip Wolfe. 1969. Convergence conditions for ascent methods. *SIAM review* 11, 2 (1969), 226–235.

Stephen J Wright. 2015. Coordinate descent algorithms. *Mathematical programming* 151, 1 (2015), 3–34.

Botao Wu, Zhendong Wang, and Huamin Wang. 2022. A GPU-based multilevel additive schwarz preconditioner for cloth and deformable body simulation. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–14.

Longhua Wu, Botao Wu, Yin Yang, and Huamin Wang. 2020. A safe and fast repulsion method for GPU-based cloth self collisions. *ACM Transactions on Graphics (TOG)* 40, 1 (2020), 1–18.

Xiaofeng Wu, Rajaditya Mukherjee, and Huamin Wang. 2015. A unified approach for subspace simulation of deformable bodies in multiple domains. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–9.

Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A scalable galerkin multigrid method for real-time simulation of deformable objects. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.

Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. 2015. Expediting precomputation for reduced deformable simulation. *ACM Trans. Graph* 34, 6 (2015).

Yin Yang, Weiwei Xu, Xiaohu Guo, Kun Zhou, and Baining Guo. 2013. Boundary-aware multidomain subspace deformation. *IEEE transactions on visualization and computer graphics* 19, 10 (2013), 1633–1645.

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*. 1–12.

Jiayi Eris Zhang, Doug James, and Danny M Kaufman. 2024. Progressive Dynamics for Cloth and Shell Animation. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–18.

Yili Zhao and Jernej Barbič. 2013. Interactive authoring of simulation-ready plants. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph. (TOG)* 29, 2 (2010), 16.

Zeshun Zong, Xuan Li, Minchen Li, Maurizio M Chiaramonte, Wojciech Matusik, Eitan Grinspun, Kevin Carlberg, Chenfanfu Jiang, and Peter Yichen Chen. 2023. Neural stress fields for reduced-order elastoplasticity and fracture. In *SIGGRAPH Asia 2023 Conference Papers*. 1–11.