Periodic Anderson Acceleration for GPU-based Elastic Body Simulation

YOUYI YANG, University of Houston, USA, USA
NIKHIL NAVKAR, Hamad Medical Corporation, Qatar, Qatar
ZHIGANG DENG, University of Houston, USA

Vertex Block Descent (VBD) solves implicit time integration through parallel per-vertex Newton steps, offering unconditional stability and GPU efficiency for elastic body simulation. Because its block Gauss-Seidel update propagates information only locally, low-frequency error modes spanning the mesh decay slowly across iterations. Chebyshev acceleration improves asymptotic convergence, but does not directly eliminate the globally correlated residuals. We introduce *periodic Anderson Acceleration* (PAA) for VBD, where a small least-squares mixing step is applied every K iterations between Chebyshev-accelerated sweeps. The two accelerators are complementary: Chebyshev drives fast local convergence while Anderson Acceleration (AA) corrects global error modes through cross-vertex coupling extracted from recent iterates. A fused GPU kernel computes the full Gram system in a single pass, limiting overhead to under 5%. We show that periodic application with a window of m=2 outperforms every-iteration AA, as Chebyshev-accelerated iterates sampled at intervals provide a more linearly independent mixing basis. Combined with per-tet stiffness ramping, PAA reduces the final elastic energy by 6-49% over Chebyshev-only VBD at less than 5% wall-clock overhead, with the largest gains on geometrically complex meshes and extreme deformations. PAA requires no global matrix assembly, no mesh hierarchy, and adds a single integer parameter.

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

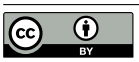
Additional Key Words and Phrases: physics-based simulation, elastic body simulation, GPU acceleration, periodic Anderson acceleration

ACM Reference Format:
Youyi Yang, Nikhil Navkar, and Zhigang Deng. 2026. Periodic Anderson Acceleration for GPU-based Elastic Body Simulation. *Proc. ACM Comput. Graph. Interact. Tech.* 9, 4, Article 61 (July 2026), 20 pages. https://doi.org/10.1145/3819834

1 Introduction

Physics-based simulation of deformable solids is a cornerstone of interactive applications in games, surgical training, and digital content creation. Modern applications increasingly demand high mesh resolutions for visual fidelity and stiff materials for realistic behavior, placing stringent demands on solver throughput. Meeting these demands within a real-time frame budget requires algorithms that exploit GPU parallelism, as sequential global solvers cannot scale with growing problem sizes. The fundamental challenge lies in designing solvers that achieve rapid convergence while remaining amenable to massively parallel execution. In practice, a faster-converging solver minimizes the elastic energy gap within a fixed frame budget. For real-time graphics, this prevents material drift on stiff, hand-animated characters; for interactive surgical simulations, it ensures more precise force-feedback responses.

Authors' Contact Information: Youyi Yang, Computer Science, University of Houston, USA, Houston, Texas, USA, youyiyang152@gmail.com; Nikhil Navkar, Hamad Medical Corporation, Qatar, Doha, Qatar, NNavkar@hamad.qa; Zhigang Deng, Computer Science, University of Houston, Houston, Texas, USA, zdeng4@central.uh.edu.



ACM 2577-6193/2026/7-ART61
https://doi.org/10.1145/3819834

Convergence speed and parallelism are often seen as inherently conflicting objectives in iterative solvers for elastodynamics [Lan et al. 2025]. Methods with fast convergence, such as Newton's method with sparse direct solves [Hecht et al. 2012], require global communication that resists efficient GPU parallelization. Conversely, highly parallel local solvers like XPBD [Macklin et al. 2016] and Jacobi-preconditioned descent [Wang and Yang 2016] plateau quickly because each iteration only propagates information across a small local neighborhood. Projective Dynamics [Bouaziz et al. 2014] partially bridges this gap with a prefactored global step, but the factorization cost and sequential global solve limit its GPU scalability. While multigrid approaches [Li et al. 2025; Lu et al. 2025] address low-frequency convergence through coarse-level corrections, they introduce significant hierarchy construction overhead and implementation complexity. A lightweight acceleration mechanism that provides global coupling without these costs remains a critical need.

Vertex Block Descent [Chen et al. 2024] (VBD) strikes a compelling balance in this landscape. By formulating the implicit Euler time step as per-vertex energy minimization, VBD reduces each local subproblem to a 3×3 Newton system that can be solved analytically. The key insight is that minimizing the local energy G_i at each vertex i guarantees a reduction in the global energy G, providing unconditional stability without line search. Parallelization is achieved through vertex graph coloring, which yields an order of magnitude fewer colors than element-based XPBD coloring [Chen et al. 2024]. Yet, VBD remains a block Gauss-Seidel method at its core: each iteration moves information by at most one vertex hop through the colored sweep. Consequently, convergence slows markedly on stiff or spatially extended meshes where global coupling is dominant.

We observe that VBD's color-by-color sweep defines a fixed-point iteration $\mathbf{x}^{k+1} = \Phi(\mathbf{x}^k)$, and that its convergence bottleneck is precisely the local nature of Φ. Anderson Acceleration (AA) [Anderson 1965; Walker and Ni 2011] is a classical technique for accelerating fixed-point iterations by mixing recent iterates through a small least-squares problem, effectively building a low-rank approximation of the inverse Jacobian without ever forming or factoring any global matrix. Peng et al. [2018] applied AA to projective dynamics and ADMM-based solvers; we find that it is equally effective for VBD's nonlinear Gauss-Seidel structure. A naive application of AA at every iteration conflicts with Chebyshev acceleration, since both extrapolate the iterate sequence, and the per-iteration overhead negates any convergence benefit.

Our key insight is that this conflict is resolved by introducing *Periodic Anderson Acceleration* (PAA): running K=16 Chebyshev-accelerated VBD sweeps, then performing one AA mixing step with a window of m=2 stored iterates. This is composition, not alternation: Chebyshev drives fast local convergence while AA periodically corrects global error modes. For high-stiffness scenarios (10× and above), we optionally integrate per-element adaptive stiffness ramping, inspired by AVBD [Giles et al. 2025]. Each element's effective stiffness ramps proportionally to its local deformation, limiting the condition number seen by early iterations. The two techniques target different bottlenecks: PAA corrects global error modes while stiffness ramping accelerates local Newton convergence on stiff elements. At low stiffness scenarios, stiffness ramping is unnecessary, and periodic AA alone is sufficient.

Our work makes the following contributions. (1) We demonstrate that VBD's iterative sweep is a fixed-point map amenable to Anderson Acceleration and show that PAA composes with Chebyshev acceleration rather than replacing it. A fused GPU kernel computes the full m×m Gram system in a single pass, keeping amortized overhead below 5%. This composition reduces the final elastic energy by 6-49% over standard VBD+Chebyshev at comparable wall-clock cost. (2) We characterize the regimes where PAA is most effective through evaluations on beam bending, nose-pull deformation, and flatten recovery across multiple mesh geometries and stiffness levels.

2 Related Work

Implicit time integration and global solvers. Implicit time integrators are widely accepted as the primary methods for simulating elastic bodies due to their exceptional stability with stiff problems. Among these, backward Euler [Baraff and Witkin 1998] is the most commonly utilized, though implicit Newmark [Hughes 2003] and BDF2 methods have also been explored. Reformulating backward Euler as an optimization problem [Gast et al. 2015; Martin et al. 2011] allows the use of large time steps and connects the simulation pipeline to well-studied optimization techniques. To circumvent a full linear solve for every Newton step, Cholesky factorization [Hecht et al. 2012] and its lagged variants reuse prefactored matrices across steps. Positive-definite projection of the Hessian [Teran et al. 2005] improves robustness, and quasi-Newton methods [Liu et al. 2017] approximate the inverse Hessian to reduce per-iteration cost. Simplifications of the underlying material model, such as stiffness warping [Müller et al. 2002] for handling rotational deformations, can reduce per-step cost but trade physical accuracy. These global methods achieve fast convergence but require global communication (e.g., sparse matrix assembly, factorization, or SpMV), making them difficult to map efficiently onto GPU hardware.

Constraint-based methods. Position Based Dynamics (PBD) [Müller et al. 2007] and its extended variant XPBD [Macklin et al. 2016] convert forces into soft constraints and directly update positions with Gauss-Seidel iterations, yielding a massively parallel solver loop. Projective Dynamics [Bouaziz et al. 2014] treats the elasticity energy as a sum of quadratic constraints, enabling a prefactored global step combined with parallel local projections; this idea can be combined with model reduction [Brandt et al. 2018] for further speedups. Parallelization of XPBD is achieved by graph coloring the constraints [Fratarcangeli and Pellacini 2015; Fratarcangeli et al. 2016]; however, the dual graph of tetrahedra has a high valence, leading to 60+ colors and correspondingly many kernel launches. Macklin et al. [2019] showed that taking many small substeps with a single solver iteration each reduces numerical damping and improves visual quality. Ton-That et al. [2023] coupled hydrostatic and deviatoric Neo-Hookean constraints into block solves with supernodal graph clustering, reducing the color count from ∼60 to ∼20. Li et al. [2025] applied algebraic multigrid preconditioning to XPBD's global system (MGPBD), achieving a faster convergence on large meshes at the cost of hierarchy construction and sparse matrix assembly. In the per-constraint Gauss-Seidel formulation, the compliance-weighted system is nearly diagonal, which can limit the effectiveness of global preconditioners.

Energy-based local solvers and GPU methods. The advent of GPGPU has driven algorithmic redesign from global linear solves to iterative parallel procedures [Bender et al. 2017]. Wang and Yang [2016] proposed Jacobi-preconditioned gradient descent with Chebyshev acceleration [Wang 2015] and backtracking line search, demonstrating that fully element-parallel solvers can be competitive with global methods. Chen et al. [2024] introduced Vertex Block Descent (VBD), replacing element-level projections with per-vertex 3×3 Newton solves that guarantee energy descent and require an order of magnitude fewer colors than element coloring. Lan et al. [2022] combined multiple Jacobi iterations into a single aggregated iteration (A-Jacobi), and later proposed JGS2 [Lan et al. 2025], which analyzes the overshoot and undershoot behavior of local solves and derives a local perturbation subspace method achieving near second-order convergence while maintaining GPU parallelism. Our work accelerates VBD from the complementary direction of iterate-level extrapolation rather than modifying the local solve itself.

Multigrid and multi-resolution methods. Multi-resolution [Capell et al. 2002; Grinspun et al. 2002] and multigrid [Bolz et al. 2003; Tamstorf et al. 2015; Zhu et al. 2010] solvers project fine-grid residual errors onto coarser grids, on which linear or nonlinear iterations are more effective at

resolving low-frequency modes. Lu et al. [2025] presented the first practical Galerkin multigrid for unstructured tetrahedral meshes using a piecewise-constant hierarchy with matrix-free coarse-to-fine sweeps. Li et al. [2025] used algebraic multigrid within a dual-space XPBD formulation. Wu et al. [2022] employed a multigrid-like preconditioner to improve convergence of GPU iterations. These approaches add hierarchy construction overhead and implementation complexity; our method avoids both by operating directly on VBD's iterate sequence.

Acceleration techniques. Chebyshev semi-iterative acceleration [Wang 2015] adds momentum to iterative solvers, requiring an estimated spectral radius ρ and providing 10-30% improvement in practice. Model reduction constructs a mapping from a low-dimensional deformation space to a full space, enabling real-time performance at the cost of precomputation and reduced generality [An et al. 2008; Barbic and James 2005]; however, these methods do not lend themselves well to varying boundary conditions and topological changes. Anderson acceleration [Anderson 1965] was originally proposed for integral equations and later analyzed as a multisecant quasi-Newton method by Walker and Ni [Walker and Ni 2011]. Peng et al. [2018] applied AA to geometry optimization and physics simulation, demonstrating acceleration of projective dynamics and ADMM-based solvers. Giles et al. [2025] introduced Augmented VBD (AVBD), combining augmented Lagrangian dual updates with adaptive stiffness ramping for hard constraints and high stiffness ratios; their stiffness ramping technique for finite-stiffness forces (Eq. 16 in their paper) is the basis for our per-element adaptation. Our approach applies AA periodically (K=16, m=2) to VBD's primal iterate sequence, composing it with Chebyshev acceleration rather than replacing it, and optionally integrates AVBD's finite-stiffness ramping for high-stiffness scenarios.

3 Background: Vertex Block Descent

We briefly review the VBD formulation [Chen et al. 2024]. Given positions $\mathbf{x}^t$ and velocities $\mathbf{v}^t$ at time t, the next-step positions $\mathbf{x}^{t+1}$ minimize the incremental potential [Martin et al. 2011]:

$$\mathbf{x}^{t+1} = \underset{\mathbf{x}}{\operatorname{argmin}}\, G(\mathbf{x}), \tag{1}$$

where the variational energy G combines inertia and elasticity:

$$G(\mathbf{x}) = \frac{1}{2h^2}\|\mathbf{x} - \mathbf{y}\|_M^2 + E(\mathbf{x}), \tag{2}$$

with inertial target $\mathbf{y} = \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}}$, mass matrix M, and total elastic potential $E(\mathbf{x})$.

VBD decomposes the global problem into per-vertex subproblems. For vertex i, the local energy gathers all force elements $\mathcal{F}_i$ incident on i:

$$G_i(\mathbf{x}) = \frac{m_i}{2h^2}\|\mathbf{x}_i - \mathbf{y}_i\|^2 + \sum_{j\in\mathcal{F}_i} E_j(\mathbf{x}). \tag{3}$$

Each vertex position is updated by a single Newton step that solves the 3×3 linear system $\mathbf{H}_i\Delta\mathbf{x}_i = \mathbf{f}_i$, where

$$\mathbf{f}_i = -\frac{m_i}{h^2}(\mathbf{x}_i - \mathbf{y}_i) - \sum_{j\in\mathcal{F}_i}\frac{\partial E_j}{\partial \mathbf{x}_i}, \quad \mathbf{H}_i = \frac{m_i}{h^2}\mathbf{I} + \sum_{j\in\mathcal{F}_i}\frac{\partial^2 E_j}{\partial \mathbf{x}_i^2}. \tag{4}$$

Vertices are partitioned by graph coloring such that no two vertices sharing a force element receive the same color; all vertices of a given color can be updated in parallel. One full sweep through all colors constitutes one VBD iteration. Note that $G(\mathbf{x}) \neq \sum_i G_i(\mathbf{x})$, since force elements appear in multiple local energies. However, when only vertex i moves, the reduction in G_i equals the resulting reduction in G, which is why VBD guarantees global energy descent without a line search. VBD requires an order of magnitude fewer colors than element-based XPBD [Chen et al.

2024]; on our test meshes, vertex coloring produces 10-13 colors compared to 60+ for element coloring.

VBD optionally applies Chebyshev semi-iterative acceleration [Wang and Yang 2016] after each sweep:

$$\mathbf{x}^{(k)} = \omega_k(\hat{\mathbf{x}}^{(k)} - \mathbf{x}^{(k-2)}) + \mathbf{x}^{(k-2)}, \quad \omega_k = \frac{4}{4 - \rho^2\omega_{k-1}}, \tag{5}$$

where $\hat{\mathbf{x}}^{(k)}$ is the sweep output and $\mathbf{x}^{(k-2)}$ is the position two iterations prior (the three-term recurrence form from Wang [2015]), $\rho \in (0, 1)$ is an estimated spectral radius, and $\omega_1 = 1$, $\omega_2 = 2/(2 - \rho^2)$. Chebyshev acceleration adds momentum along the recent iterate trajectory, making it the standard acceleration technique for VBD.

Despite its strengths, VBD is a first-order block Gauss-Seidel method: each iteration propagates information by at most one vertex hop through the colored sweep. On spatially extended or stiff meshes, this local propagation creates a convergence bottleneck, as a global equilibrium requires many iterations to establish. Chebyshev acceleration adds momentum that partially mitigates this, but it cannot introduce coupling between vertices that have not yet communicated through the sweep structure. This motivates the search for a mechanism that provides global coupling without the cost of a global linear solve or multigrid hierarchy.

4 Method

4.1 Anderson Acceleration for VBD

We observe that one complete VBD sweep (iterating over all color groups) defines a nonlinear map $\Phi : \mathbb{R}^{3N} \to \mathbb{R}^{3N}$ from the current iterate to the next:

$$\mathbf{x}^{k+1} = \Phi(\mathbf{x}^k). \tag{6}$$

The converged solution $\mathbf{x}^*$ satisfies $\mathbf{x}^* = \Phi(\mathbf{x}^*)$, making VBD a fixed-point iteration. The map Φ is nonlinear because each vertex's Newton step depends on the current positions of its neighbors, which change during the sweep.

Anderson acceleration (AA) [Anderson 1965; Walker and Ni 2011] accelerates fixed-point iterations by mixing recent iterates. At iteration k, we define the residual $\mathbf{r}^k = \Phi(\mathbf{x}^k) - \mathbf{x}^k$ and store the most recent m+1 pairs $\{(\mathbf{x}^{k-m}, \mathbf{r}^{k-m}), \ldots, (\mathbf{x}^k, \mathbf{r}^k)\}$. The mixing coefficients $\alpha \in \mathbb{R}^{m+1}$ are determined by minimizing the combined residual:

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}} \left\| \sum_{i=0}^{m} \alpha_i \mathbf{r}^{k-i} \right\|^2 \quad \text{s.t.} \quad \sum_{i=0}^{m} \alpha_i = 1. \tag{7}$$

The accelerated iterate is then the corresponding weighted combination of the swept positions:

$$\mathbf{x}^{k+1} = \sum_{i=0}^{m} \alpha_i^* \, \Phi(\mathbf{x}^{k-i}). \tag{8}$$

Walker and Ni [Walker and Ni 2011] showed that AA is equivalent to a multisecant quasi-Newton method: the mixing coefficients implicitly construct a low-rank approximation of the inverse Jacobian $(I - \nabla\Phi)^{-1}$ from finite differences of recent iterates. In other words, AA extracts second-order information about the global system from the history of first-order VBD sweeps. This means that AA-m captures global coupling information that the local VBD sweep misses, effectively extending the information radius from one vertex hop to the entire mesh, without constructing any hierarchy or global matrix. The rank-m approximation is coarse but sufficient: on the meshes we tested, m=2 captures the dominant mode of the residual trajectory while keeping the Gram system a well-conditioned 2×2 matrix solvable by direct inversion on the GPU.

The constrained least-squares in Equation (7) reduces to an unconstrained $m \times m$ system. Define $\Delta\mathbf{r}^j = \mathbf{r}^{k-j+1} - \mathbf{r}^{k-j}$ for $j = 1, \ldots, m$, and form the Gram matrix $[\mathbf{R}]_{ij} = \langle \Delta\mathbf{r}^i, \Delta\mathbf{r}^j \rangle$ and the right-hand side $\mathbf{b}_j = \langle \Delta\mathbf{r}^j, \mathbf{r}^k \rangle$. The coefficients $\theta \in \mathbb{R}^m$ satisfy $\mathbf{R}\theta = \mathbf{b}$, and the mixing weights follow as $\alpha_0 = 1 - \sum \theta_j$, $\alpha_j = \theta_j - \theta_{j+1}$. For m=2, this is a 2×2 system solvable by direct inversion.

Applying AA at every VBD iteration conflicts with Chebyshev acceleration, since both modify the iterate trajectory. We resolve this by introducing PAA: every K iterations, one AA mixing step replaces the Chebyshev extrapolation, using the last m+1 stored iterates. Between AA steps, standard Chebyshev acceleration (Equation (5)) is applied, and the Chebyshev recurrence is reset ($\omega \leftarrow 1$) after each AA step. The key insight is that Chebyshev-accelerated iterates, sampled every K steps, are more diverse than raw VBD outputs sampled every step: the polynomial momentum produces iterates that explore different directions in the residual space, improving the conditioning of AA's least-squares problem. We use K=16 throughout; with $\rho \approx 0.93$, sixteen Chebyshev steps approximately complete one full cycle of the dominant spectral mode, so iterates sampled at this interval span the most informative directions for AA's low-rank correction. The choice of m=2 keeps the Gram system a well-conditioned 2×2 matrix; larger m adds nearly collinear residual differences that degrade conditioning without improving the correction (Section 5.5).

The values of m and K trade-off along three axes. (i) Convergence: a VBD sweep's residual is dominated by a few global modes, so m=2 (whereas m=1 reduces to single-direction damping) already spans the useful subspace, while an update period of one spectral cycle maximizes new information per correction. (ii) Stability: Setting m=2 reduces the Gram matrix solver to a closed-form 2×2 inverse, completely avoiding the need for explicit regularization. (iii) Efficiency: AA storage scales as $O(mN)$ and amortized cost as $1/K$, therefore, a small history window combined with a long period represents the most computationally efficient configuration. Ultimately, larger m or smaller K simultaneously degrades all three performance metrics (Section 5.5).

We implemented AA entirely on the GPU. The m+1 residual vectors and iterate history are stored as GPU buffers, requiring $(m+1) \times 3N$ additional floats. A single fused kernel computes all $m(m+1)/2$ Gram matrix entries and m RHS entries simultaneously via atomic reductions over vertices, replacing $m(m+1)/2 + m$ separate kernel launches with one. For m=2, this means 3 unique Gram entries and 2 RHS entries. The 2×2 system is solved in a single-thread kernel via direct inversion, and the final mixing (Equation (8)) is a parallel weighted sum. One device barrier separates the sweep from the one-pass atomic Gram reduction; the 2×2 solve floors its determinant to skip near-collinear cases (did not fire in our tests).

Optional stiffness ramping for high-stiffness scenarios. At high stiffness, the hydrostatic term (λ) produces steep gradients that create non-smooth iterate trajectories, degrading AA's least-squares mixing quality. Following the finite-stiffness ramping of AVBD [Giles et al. 2025] (Eq. 16 in their paper), we optionally maintain per-element effective Lamé parameters that ramp toward target values proportionally to local deformation:

$$\mu_{\text{eff}}^{(k+1)}[t] = \min\big(\mu,\ \mu_{\text{eff}}^{(k)}[t] + \beta\,\mu\,C_{\text{dev}}[t]\big), \tag{9}$$

where $C_{\text{dev}}[t] = \sqrt{\max(0, \|F[t]\|^2 - 3)}$ measures deviatoric strain; an analogous update applies to λ_{eff} using volumetric strain $C_{\text{vol}}[t] = |J[t] - 1|$. Between time steps, effective stiffness is warm-started with exponential decay: $\mu_{\text{eff}}^{(0)}[t] = \max(r_0\,\mu,\ \gamma\,\mu_{\text{eff}}^{\text{prev}}[t])$ with γ=0.99, r_0=0.01. The only modification to VBD's local solver is replacing constant (μ, λ) with per-element $(\mu_{\text{eff}}[t], \lambda_{\text{eff}}[t])$ in the PK1 stress and Hessian, adding negligible overhead. Stiffness ramping and PAA target different bottlenecks: ramping smooths the stiffness landscape for early iterations while AA corrects global

Algorithm 1 VBD with Periodic Anderson Acceleration (PAA)

Require: $\mathbf{x}^t, \mathbf{v}^t, \mathbf{a}_{\text{ext}}$, material params (μ, λ), period K=16, window m=2
Ensure: $\mathbf{x}^{t+1}, \mathbf{v}^{t+1}$

1: $\mathbf{y} \leftarrow \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}}$ ▷ Inertial target
2: $\mathbf{x} \leftarrow$ initial guess; $\omega \leftarrow 1$; $n_{\text{aa}} \leftarrow 0$; $k_{\text{last}} \leftarrow 0$ ▷ Init
3: Optionally warm-start $\mu_{\text{eff}}[t], \lambda_{\text{eff}}[t]$ ▷ AVBD only
4: **for** $k = 1$ **to** $n_{\max}$ **do**
5: Optionally update $\mu_{\text{eff}}[t], \lambda_{\text{eff}}[t]$ (Eq. 9) ▷ AVBD only
6: **for** each color c **do** ▷ VBD sweep
7: **parallel for** vertex i in color c:
8: Compute $\mathbf{f}_i, \mathbf{H}_i$ ((4))
9: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{H}_i^{-1}\mathbf{f}_i$ ▷ 3×3 Newton step
10: **end for**
11: **if** $k - k_{\text{last}} \geq K$ **or** $n_{\text{aa}} < m{+}1$ **then** ▷ Periodic AA (or buffer fill)
12: Store iterate $\mathbf{x}^k$ and residual $\mathbf{r}^k = \Phi(\mathbf{x}^k) - \mathbf{x}^k$; n_{aa} += 1
13: **if** $n_{\text{aa}} \geq 2$ **then**
14: Compute $m{\times}m$ Gram via fused kernel; solve ((7))
15: $\mathbf{x} \leftarrow \sum \alpha_i^* \mathbf{x}_{\text{stored}}^{(i)}$ ((8))
16: **end if**
17: $k_{\text{last}} \leftarrow k$; $\omega \leftarrow 1$ ▷ Reset Chebyshev
18: **else** ▷ Chebyshev acceleration
19: Apply (5); update ω
20: **end if**
21: **end for**
22: $\mathbf{v}^{t+1} \leftarrow (\mathbf{x} - \mathbf{x}^t)/h$

error modes. At low stiffness (1×-3×), ramping is unnecessary and VBD+PAA alone achieves the best results; at high stiffness (10× and above), combining both (AVBD+PAA) is beneficial (Section 5.2).

4.2 Algorithm Summary

Algorithm 1 maps line by line to our method. Line 1 forms the inertial target. The inner sweep, lines 6-9, is exactly baseline VBD, unchanged by our method. Line 11 is the sole added branch: once K iterations have elapsed since the last firing (or the history is still filling), the Chebyshev step is replaced by AA, which solves the fused Gram system and mixes the stored iterates (lines 14-15); otherwise the standard Chebyshev extrapolation runs (line 19). Line 22 recovers the velocity. With K=16 and 100 iterations, AA fires eight times per step: two warm-up mixes at k=2, 3 and six K-periodic mixes at iterations 19, 35, 51, 67, 83, and 99. Mesh connectivity enters only via the force set $\mathcal{F}_i$ and the coloring, identical for 2D triangle and 3D tetrahedral meshes.

Collision detection and response follow the standard VBD procedure [Chen et al. 2024]: discrete collision detection at the start of each time step, continuous collision detection every n_{col} iterations, and quadratic penalty energies folded into the local vertex system (Equation (4)). AA mixing is applied uniformly to all vertex positions; in the presence of contact, the mixed iterate may not satisfy collision constraints, and subsequent VBD sweeps must resolve any resulting penetrations.

We use m=2 for the AA window, K=16 for the AA period, ρ=0.93 for Chebyshev, and optionally β=0.1 for the stiffness ramping rate. Only m and K are new parameters introduced by this work. The spectral radius ρ is taken directly from the VBD paper [Chen et al. 2024] (Table 3 in [Chen et al.

Table 1. Test meshes used in our evaluation. Material parameters follow VBD and JGS2 conventions.

Mesh	#Verts	#Tets	μ	λ
Beam	692	3,482	10^5	10^6
Spot	3,781	20,556	10^5	10^6
Octopus	4,188	20,907	10^5	10^6
Armadillo	7,608	39,172	10^6	10^7
Bunny	11,592	59,588	10^5	10^6
Squirrel	15,864	73,146	10^5	10^6

Table 2. Simulation parameters. All scenes use stable Neo-Hookean [Smith et al. 2018] with ρ=100 kg/m^3, ν=0.45. Beam stiffness levels 3×/10×/30×/100× scale μ,λ proportionally; ρ_s = 0.86/0.93/0.95/0.97 following [Chen et al. 2024].

Scene	Mesh	Simulation						Material	
		S	n_{iter}	h (s)	ρ_s	K	m	μ	λ
Beam bending (1×)	Beam	5	100	1/60	0.75	16	2	1×10^5	1×10^6
Nose pull	Armadillo	1	100	1/300	0.93	16	2	1×10^6	1×10^7
Flatten recovery	Various	1	100	1/60	0.93	16	2	1×10^6	1×10^7

2024], tuned per stiffness level: 0.75/0.86/0.93/0.95/0.97 at 1×/3×/10×/30×/100×). The ramping rate β=0.1 is selected via sweep on beam 10× ($\beta \in \{0.01, 0.05, 0.1, 0.5, 1.0, 10.0\}$, Section 5.2); the AVBD work [Giles et al. 2025] defaults to β=10 without Chebyshev, which saturates the ramp too quickly when combined with Chebyshev's 100-iteration budget. These parameters are held constant across all test meshes and scenarios unless otherwise noted. Section 5.5 validates these empirical choices.

5 Results

5.1 Experimental Setup

We evaluated our method on a suite of tetrahedral meshes spanning a range of vertex counts and geometric complexity (Table 1). All meshes use the stable Neo-Hookean energy model [Smith et al. 2018] with Lamé parameters consistent with the VBD [Chen et al. 2024] and JGS2 [Lan et al. 2025] evaluation suites.

Three test cases were used in our evaluations (Table 2). **Case 1** (beam bending): a cantilever beam under gravity with one end fixed, tested across five stiffness levels (1×-100× reference stiffness), 100 iterations per time step, 300 time steps. **Case 2** (nose pull): an armadillo mesh with fixed fingers and toes, nose pulled instantly to full displacement at frame 1, h=1/300, 100 iterations per time step. **Case 3** (flatten recovery): meshes squeezed to 1% of original height and released, 100 iterations per time step, 60 time steps.

We compared our method against VBD [Chen et al. 2024] (no acceleration), VBD+Chebyshev (the standard VBD acceleration), and AVBD+Chebyshev (per-tet stiffness ramping [Giles et al. 2025] combined with Chebyshev acceleration, without AA). The original AVBD formulation [Giles et al. 2025] does not use Chebyshev acceleration and defaults to a ramping rate β=10; we combine it with Chebyshev and set β=0.1, chosen via sweep on beam 10× ($\beta \in \{0.01, 0.05, 0.1, 0.5, 1.0, 10.0\}$): β=0.1 is optimal for 100 iterations, ramping stiffness gradually over the full iteration budget rather

than saturating early (β=10) or too slowly (β=0.01). Our method is VBD+PAA (m=2, K=16) and optionally AVBD+PAA for high-stiffness scenarios. The primary metric is the total elastic potential energy $E(\mathbf{x}^k) = \sum_t V_0\, \psi(F_t)$ evaluated at the final iteration of each time step, where ψ is the stable Neo-Hookean energy density. We report E rather than the full variational energy G (Equation 2) because all methods start from the same inertial prediction $\mathbf{y}$ and run the same number of iterations; the inertial term is nearly identical across methods at convergence, so differences in G are dominated by differences in E. We also report per-iteration convergence curves, energy-vs-wall-clock-time plots, and visual quality comparisons.

All experiments were implemented in Taichi 1.7.4 and run on a system with an Intel Core i9-13900K CPU, 60 GB DDR5 RAM, and an NVIDIA RTX 4090 GPU with CUDA backend. Before benchmarking, two dry solver steps with full state restore and GPU synchronization are executed to eliminate JIT compilation overhead from all measured frames.

5.2 Beam Bending: Stiffness Scaling

We tested a cantilever beam (692 vertices, 3.5K tetrahedra) under gravity across five stiffness levels spanning two orders of magnitude. This benchmark, also used by Chen et al. [2024] (Figure 13 in [Chen et al. 2024]), isolates the effect of material stiffness on solver convergence: stiffer beams create stronger coupling between vertices, making the local VBD sweep less effective per iteration.

Figure 1 shows the average elastic energy for each method. At low stiffness (1×-3×), VBD+PAA achieves 30-48% lower energy than VBD+Chebyshev; at high stiffness (10×-100×), combining PAA with per-tet stiffness ramping (AVBD+PAA) reduces energy by a further 21-43% beyond AVBD+Chebyshev. The crossover occurs around 10× stiffness, where the stiffness ramp begins to contribute.

At a low stiffness, the elastic forces are weak relative to inertia, so the iterate trajectory is smooth and the Chebyshev polynomial already captures most of the convergence structure; AA's global correction provides an additional but modest improvement, while AVBD+Chebyshev's stiffness ramp is unnecessary and slightly harmful (it starts elements at reduced stiffness when they are already soft). At a high stiffness, the elastic forces dominate and create strong cross-vertex coupling that the local VBD sweep cannot resolve efficiently; both AVBD+Chebyshev (which smooths the stiffness landscape for early iterations) and AA (which provides global coupling) contribute, and their combination achieves the best results because they address different bottlenecks.

Table 3 presents the optimal update period K for each stiffness level, demonstrating that a fixed value of $K = 16$ remains within 3% of the optimal K across all configurations. As shown in the ratio columns, our approach reduces the elastic energy gap to $0.52\times - 0.79\times$ that of the best baseline (a 21% to 48% reduction) while execution time remains virtually unchanged at $1.00\times - 1.13\times$. This achieves a substantial gain in stability within the margin of standard timing noise, providing the underlying mechanism for the performance crossover observed at a 10× stiffness increase, where stiffness ramping begins to yield benefits.

Figure 2 shows frame 3 of the beam at three stiffness levels (10×, 30×, 100×), comparing AVBD+Chebyshev, AVBD+PAA (ours), and the Newton-converged ground truth, with per-substep timing reported in the legend. At 10× stiffness, the baseline (AVBD+Chebyshev) visibly under-estimates the sag at the free end, while our method (AVBD+PAA) closely matches the ground truth. At 30× and 100×, the separation narrows as all methods approach the stiff limit, but our method remains the closest to the ground truth.

A natural concern is whether the convergence improvement justifies any additional per-step cost. Figure 3 addresses this by plotting elastic energy against wall-clock time (right) alongside

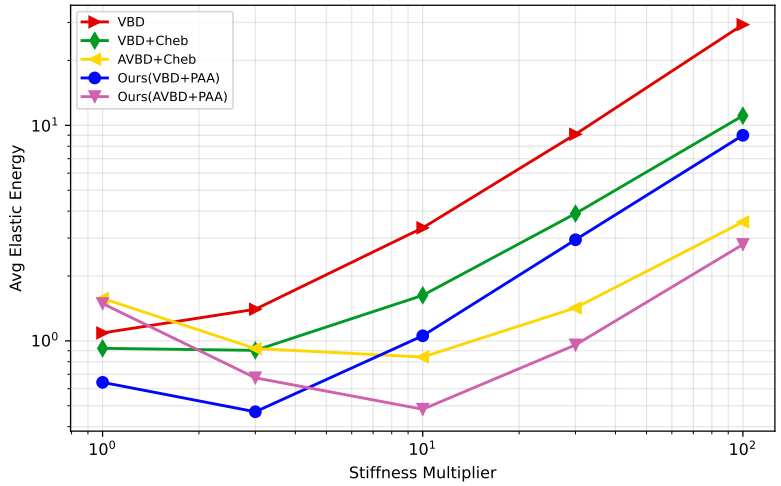


Fig. 1. **Stiffness scaling.** Average elastic energy on beam bending at five stiffness levels. VBD+PAA (ours) is lowest at 1×-3×; AVBD+PAA (ours) is lowest at 10×-100×. All PAA variants beat VBD+Chebyshev across the full range.

Table 3. Experimental result summary of Beam bending (m=2, K=16, 100 iterations). Elastic energy and time are averaged over 300 time steps. The best baseline differs by stiffness: VBD+PAA at 1×-3× (where AVBD+Chebyshev hurts), AVBD+Chebyshev at 10×-100× (where stiffness ramping helps). Ratios below 1.0 indicate our method improves over the respective baseline.

		Our best		vs best baseline	
Stiffness	Method	Energy	Time	Energy	Time
1×	VBD+PAA	6.4×10^{-1}	50.8 ms	0.70×	1.02×
3×	VBD+PAA	4.7×10^{-1}	52.5 ms	0.52×	1.06×
10×	AVBD+PAA	4.8×10^{-1}	54.5 ms	0.57×	1.00×
30×	AVBD+PAA	9.6×10^{-1}	54.0 ms	0.68×	1.04×
100×	AVBD+PAA	2.8×10^{0}	52.9 ms	0.79×	1.13×

time steps (left) for the 10× stiffness level. The oscillations in both plots reflect the beam's dynamic bending cycle: each swing creates a new nonlinear system that the solver must reconverge. AVBD+PAA consistently reaches the lowest energy troughs on both axes, confirming that the improvement is not merely per-iteration but translates to better accuracy at any given wall-clock budget. VBD+Chebyshev and plain VBD exhibit higher energy peaks and slower decay of the oscillation envelope, indicating that under-converged time steps accumulate errors that affect subsequent dynamics. Figure 4 shows the same comparison at the remaining four stiffness levels. The pattern is consistent: VBD+PAA reaches the lowest energy at low stiffness (1×, 3×), while AVBD+PAA dominates at high stiffness (30×, 100×).

5.3 Nose Pull

We pull a single nose vertex of an armadillo mesh (7.6K vertices, 39K tetrahedra) instantly outward at frame 1 (100% displacement as a step function) while fixing the fingers and toes, following the

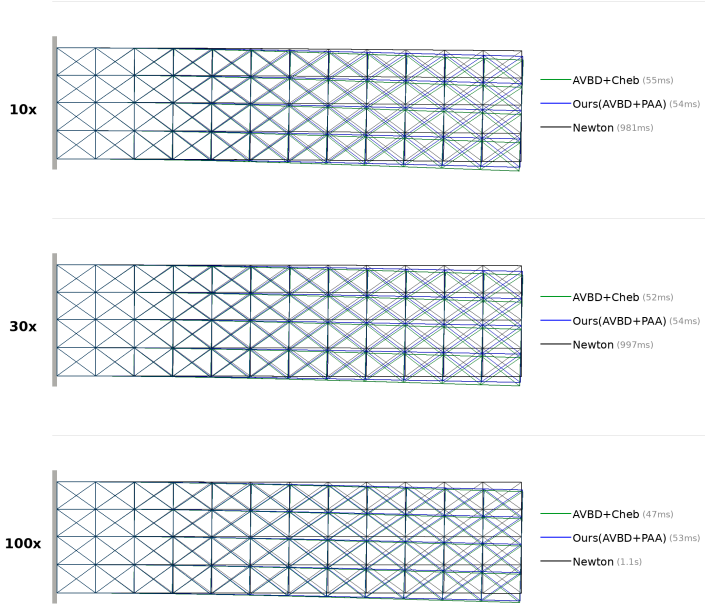


Fig. 2. **Visual quality comparison.** Beam at frame 3 for 10×, 30×, and 100× stiffness. Each row compares AVBD+Chebyshev, AVBD+PAA (ours), and the Newton ground truth, with per-substep timing in the legend. Our method tracks the Newton solution more closely than the baseline at all stiffness levels. The figure is rendered as an orthogonal side view for a clear comparison; the underlying simulation uses a 3D tetrahedral mesh.

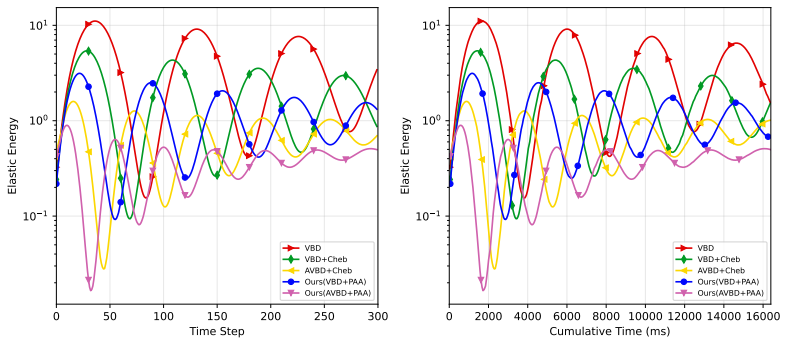


Fig. 3. **Beam** 10× **stiffness.** Elastic energy vs. time step (left) and vs. wall-clock time (right) over 600 steps. AVBD+PAA (ours) reaches the lowest energy on both axes. The oscillations reflect the beam's dynamic bending cycle.

setup of Chen et al. [2024] (i.e., Figure 16 in [Chen et al. 2024]). This scenario tests convergence under an extreme localized perturbation.

Figure 5 shows a visual comparison at frame 10. Since the perturbation originates from a single vertex, convergence depends on how quickly the solver propagates the deformation into the surrounding mesh. In VBD and VBD+Chebyshev, the torso region adjacent to the nose (red boxes)

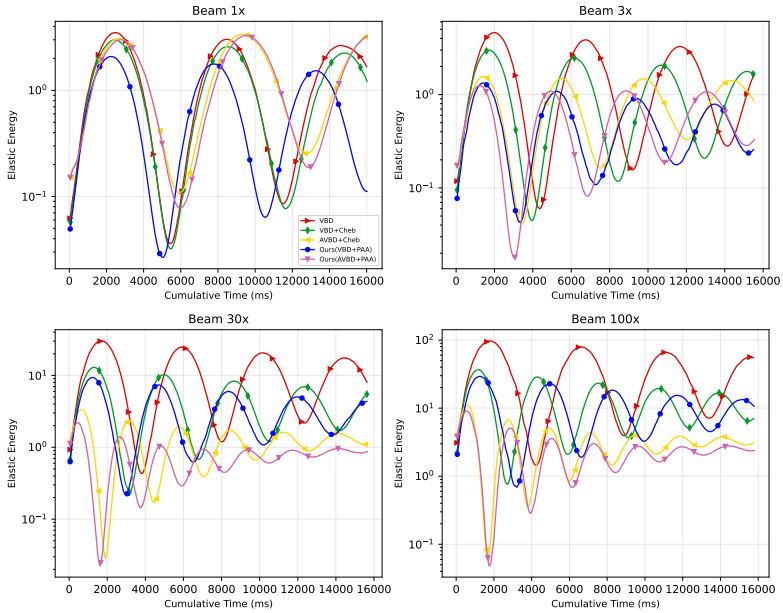


Fig. 4. **Energy vs. time at other stiffness levels.** 2×2 grid for 1×, 3×, 30×, 100× (see Figure 3 for 10×). Our method (VBD + PAA for low stiffness levels and AVBD + PAA for high stiffness levels) consistently reaches the lowest energy troughs across all stiffness levels.

remains visibly compressed, with the deformation concentrated near the pull site. Our methods propagate the effect further into the chest and shoulders, producing a more relaxed body posture; this is most apparent with K=3, where more frequent AA mixing accelerates global information transfer.

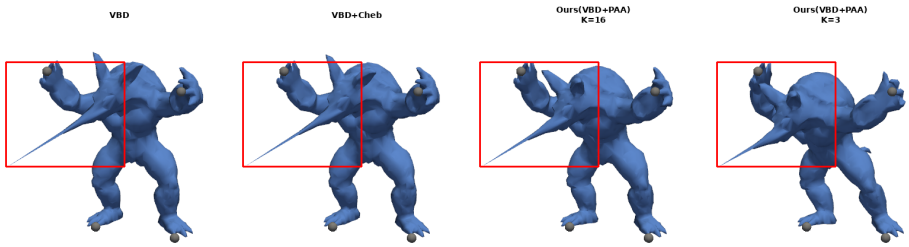
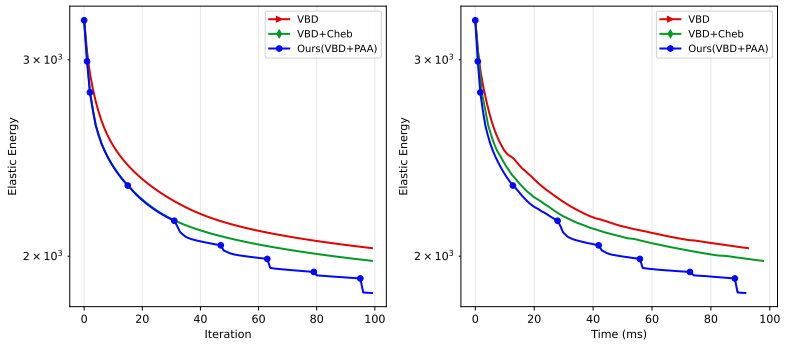


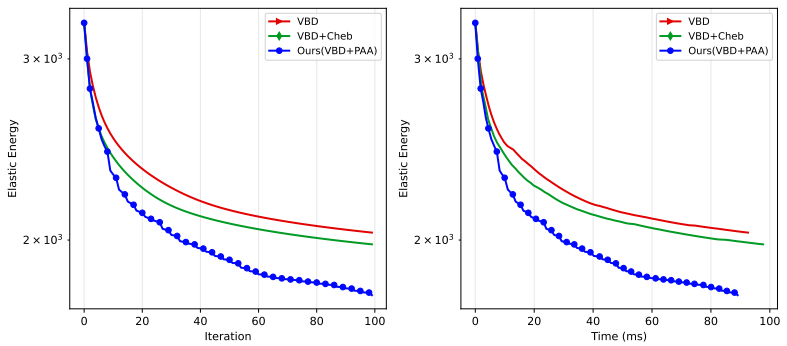
Fig. 5. **Nose pull visual comparison.** Stanford Armadillo model at frame 10 after instant nose pull. Red boxes highlight the torso region adjacent to the pull site.

Figure 6 shows per-iteration convergence at the first pull step (time step 1), where the nose suddenly jumps outward and the solver must reconverge from the perturbed state. With the default K=16, VBD+PAA reaches lower energy than VBD+Chebyshev, with the gap emerging after approximately iteration 30 (Figure 6a). The improvement is moderate because the deformation is localized to the nose region; most of the mesh is near equilibrium and does not benefit from global correction.

Setting K=3 for this scenario demonstrates that a smaller period can achieve better convergence with similar overhead (Figure 6b). The smaller period is beneficial here because the mesh has only 7.6K vertices: with fewer degrees of freedom, the AA least-squares problem is better conditioned even with frequently sampled iterates, and the overhead of more frequent PAA firings is offset by faster convergence. This illustrates that K admits per-scenario tuning analogous to the spectral radius ρ in Chebyshev acceleration, though K=16 remains a robust default across our other test cases. Figure 7 shows per-time-step energy over 10 post-pull frames for both K=16 and K=3,



(a) Default K=16.



(b) K=3: better convergence with similar overhead.

Fig. 6. **Nose pull convergence** (armadillo 40K, time step 1, 100 iterations). (a) With default K=16, VBD+PAA tracks below VBD+Chebyshev after iteration 30. (b) With K=3, VBD+PAA achieves better convergence with similar overhead.

confirming that the improvement persists across multiple time steps.

To test resolution scaling, we run the same scenario on armadillo meshes from 1.7K to 304K vertices using K=16 (Figure 8). Our method (VBD+PAA) tracks at or below VBD+Chebyshev at all resolutions, with the time overhead remaining proportional to mesh size. This confirms that the fused Gram kernel scales well: its cost grows with vertex count (one parallel reduction over all vertices), but so does the cost of the VBD sweep itself, keeping the relative overhead stable.

5.4 Flatten Recovery: Stress Test

We squeeze each mesh to 1% of its original height and release it, testing recovery from extreme deformations. This is the most challenging scenario in our evaluation: at 1% compression, the

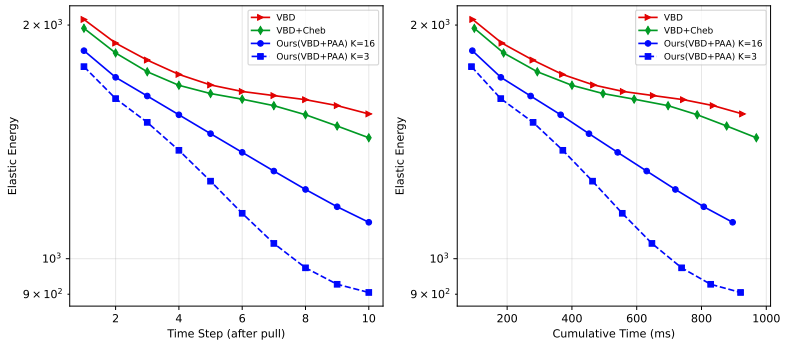


Fig. 7. **Nose pull per-step energy** (armadillo 40K, 10 post-pull frames). Elastic energy vs. time step for K=16 and K=3. VBD+PAA consistently tracks below VBD+Chebyshev.

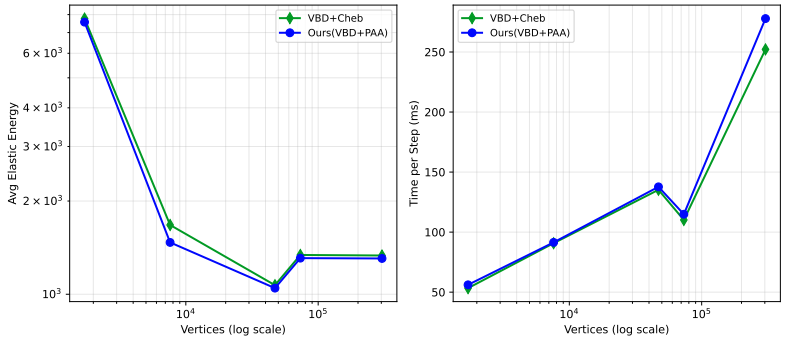


Fig. 8. **Nose pull resolution scaling.** Average energy and time per time step on armadillo meshes from 1.7K to 304K vertices (K=16). The meshes are independently generated at each resolution (not uniform refinement), which accounts for the non-monotonic energy profile. VBD+PAA matches or improves upon VBD+Chebyshev at all scales.

deformation gradient F is far from the identity, creating large elastic forces that require global coordination to resolve. The first five time steps are the critical recovery phase; after that, the mesh is nearly restored and the energy plateaus. This is precisely the regime where AA's global correction is most valuable: the elastic energy has strong low-frequency structure (all vertices need to move upward simultaneously), which the local VBD sweep propagates slowly.

Figure 9 shows the spot mesh (3.8K vertices, 21K tetrahedra) recovering from 1% compression over the first three time steps. A reference column shows the rest pose and the initial 1% compressed state. After time step 1, all three methods produce a similar blob, but differences become apparent in subsequent steps. By time step 2, VBD+PAA (bottom row) has separated the front legs (red boxes, columns 1 and 3) while VBD and VBD+Chebyshev still show them merged. By time step 3, VBD+PAA reaches a recognizable rest pose ahead of the baselines.

The per-time-step elastic energy over the full 60-step simulation (Figure 10) tells a more dramatic story. On the spot mesh, VBD+PAA reaches energy an order of magnitude below VBD+Chebyshev by time step 30, and this gap persists throughout the simulation. On sustained recovery from extreme deformations, PAA provides a qualitative improvement in convergence behavior, not just a quantitative one.

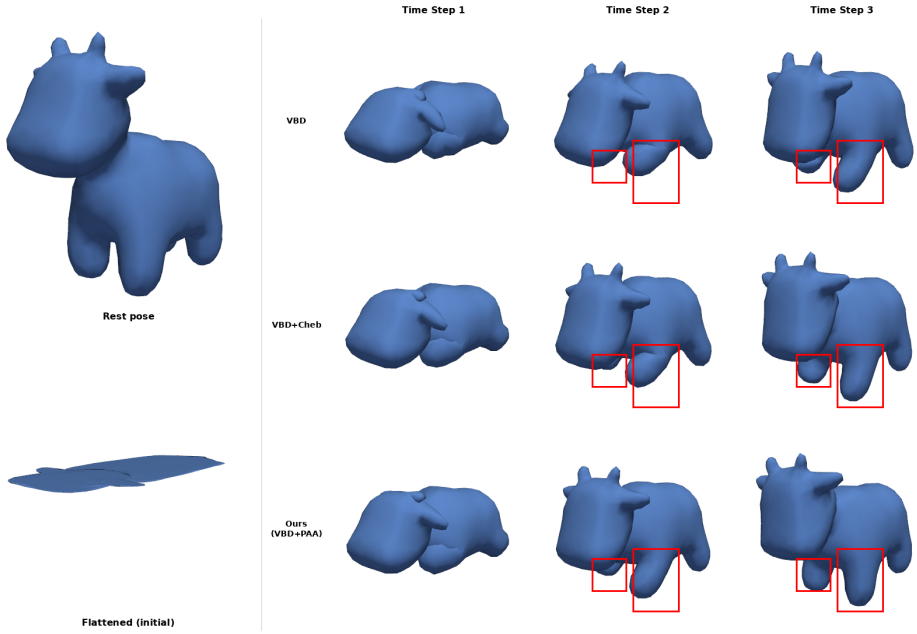


Fig. 9. **Flatten recovery.** Spot model (courtesy of Keenan Crane) squeezed to 1% height and released. Left: reference column showing rest pose and initial 1% compressed state. Right: 3×3 grid with rows VBD, VBD+Chebyshev, VBD+PAA (ours) and columns for the first three time steps. Red boxes (columns 1 and 3) highlight regions where VBD+PAA resolves fine geometric detail earlier than the baselines.

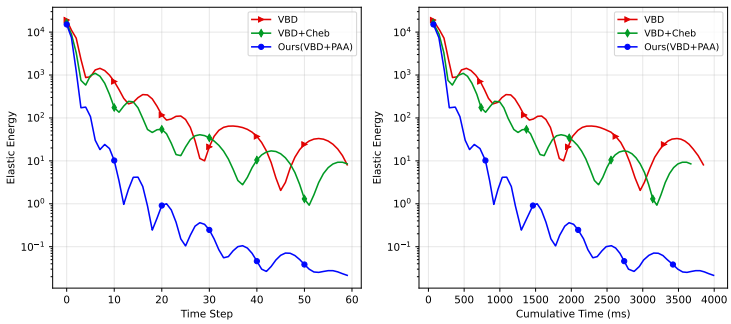


Fig. 10. **Flatten per-step energy** (spot 20K, 60 time steps). Elastic energy vs. time step (left) and vs. wall-clock time (right). On the spot mesh, VBD+PAA reaches energy an order of magnitude below VBD+Chebyshev by time step 30.

Table 4 reports the average elastic energy over time steps 1-5 across five mesh geometries. VBD+PAA with K=16 reduces elastic energy by 6-49% compared to VBD+Chebyshev on every mesh, with wall-clock overhead ranging from −5% (octopus) to +12% (spot).

The variation in improvement across meshes reflects geometric complexity. Octopus (49% improvement) has tentacles that fold into the body during compression, creating spatially separated regions that must coordinate during recovery; AA's global mixing directly addresses this by coupling the tentacle iterates with the body. Armadillo (6% improvement) is the most compact geometry

with the smallest surface-to-volume ratio, so the local VBD sweep already propagates information efficiently and the residual headroom PAA can recover is correspondingly small. Bunny and squirrel fall between these extremes, with their protruding ears and tails benefiting from global correction. The two-orders-of-magnitude difference in per-tet energy between octopus/squirrel and spot/bunny/armadillo reflects recovery speed: octopus and squirrel have thin protruding features that spring back quickly, so by time steps 1-5 they are already near rest pose with low elastic energy, whereas the bulkier meshes retain more residual deformation.

Table 4. Flatten recovery (1% squeeze, time steps 1-5 avg). PAA uses m=2, K=16. Energy is reported per tetrahedron (E/N_{tet}) for cross-mesh comparability.

			VBD+Cheb		VBD+PAA		vs Cheb	
Mesh	#Verts	#Tets	E/N_t	Time	E/N_t	Time	Energy	Time
Spot	3,781	20,556	2.91×10^{-1}	67.0 ms	2.35×10^{-1}	75.2 ms	0.81×	1.12×
Octopus	4,188	20,907	1.75×10^{-3}	77.1 ms	8.93×10^{-4}	73.6 ms	0.51×	0.95×
Armadillo	7,608	39,172	4.43×10^{-1}	92.7 ms	4.16×10^{-1}	93.1 ms	0.94×	1.00×
Bunny	11,592	59,588	6.73×10^{-1}	88.9 ms	5.02×10^{-1}	86.0 ms	0.75×	0.97×
Squirrel	15,864	73,146	1.25×10^{-3}	87.5 ms	9.15×10^{-4}	91.5 ms	0.73×	1.05×

5.5 Ablation Study

We ablate the PAA window size m and period K on the nose-pull scenario (armadillo 40K, 100 iterations, time step 1).

Window size m. Figure 11 shows elastic energy as a function of K for $m \in \{2, 3, 4\}$. m=2 dominates at every K value. m=3 provides moderate improvement at large K but is consistently worse than m=2. m=4 barely improves over VBD+Chebyshev at any K. The reason is conditioning: the $m\times m$ Gram matrix $\mathbf{R}$ is formed from inner products of residual differences, and with m=3 or m=4 these differences become nearly collinear after the first few Chebyshev-accelerated sweeps, leading to an ill-conditioned system. With m=2, the 2×2 system is trivially conditioned and the single difference vector captures the dominant correction direction. Larger m also incurs higher per-firing cost: at K=1, m=3 takes 2.4× longer per time step than m=2 (Figure 11, right panel), making it worse in both convergence and wall-clock time.

Period K. The optimal K depends on the problem. On this medium-sized mesh, smaller K values such as K=3 achieve better convergence with similar overhead (Figure 11, left). On beam bending (Table 3) and flatten recovery, K=16 is optimal. The difference reflects a trade-off: smaller K provides more frequent global corrections but interrupts Chebyshev's polynomial momentum; larger K allows Chebyshev to build longer polynomial runs between AA steps, improving asymptotic convergence. On smaller meshes where the Chebyshev polynomial converges quickly, frequent AA corrections (small K) provide more value; on stiffer or larger problems where Chebyshev needs longer runs, sparse AA (large K) is better. We recommend K=16 as the default because it is robust across our most challenging scenarios.

Fused Gram kernel. The fused Gram kernel reduces per-firing AA overhead by approximately 50% compared to launching separate reduction kernels for each Gram entry. For m=2, a single kernel computes all 3 Gram matrix entries and 2 RHS entries via atomic reductions over vertices. Combined with K=16, the amortized overhead is under 5% of the total solve time.

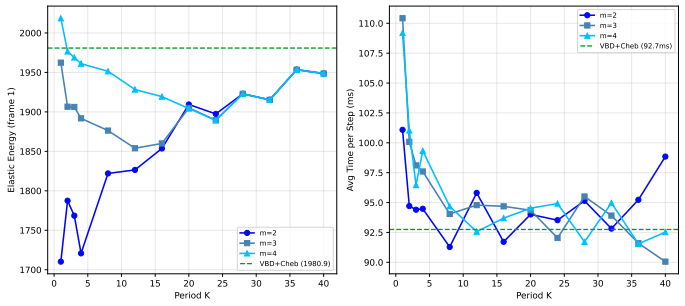


Fig. 11. **Ablation:** $K \times m$. Elastic energy (time step 1) vs. AA period K for m=2, 3, 4 on armadillo 40K nose pull. m=2 is strictly best at all K values. Dashed line: VBD+Chebyshev baseline.

5.6 Performance

Figure 12 shows the average computing time per time step on beam bending across all stiffness levels. PAA adds less than 5% overhead at most stiffness levels. On long simulations (Case 1; 300 time steps) the wall-clock overhead stabilizes to −1% to +5%; the wider −5% to +12% range on Case 3 (Table 4) reflects timing variability over the 5-step recovery phase. The matched computational cost represents a core advantage rather than a limitation. Because real-time simulations operate within a strict, fixed per-step budget, PAA achieves a significantly better-converged state at equal frame times — yielding a 6% to 49% lower elastic energy than a Chebyshev-only baseline. Equivalently, this saves the additional iterations that a Chebyshev-only solver would require to match our convergence level. Under tight budgets, this optimization delivers reduced material drift and more accurate force responses at zero additional frame cost.

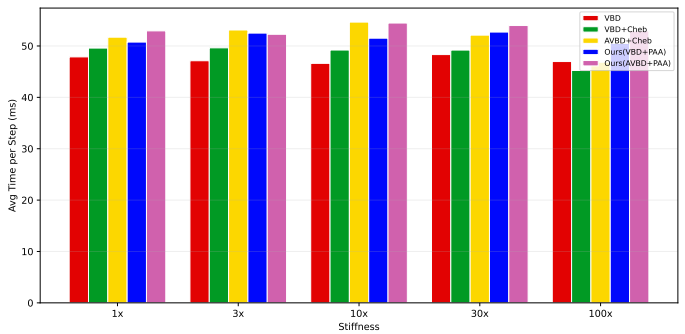


Fig. 12. **Computing time cost.** Average ms per time step on beam bending at five stiffness levels. PAA adds negligible overhead; AVBD+PAA is slightly more expensive due to the per-tet stiffness update.

6 Limitations and Future Work

Our evaluation compares PAA against VBD variants rather than fundamentally different solver families such as MGPBD [Li et al. 2025] or JGS2 [Lan et al. 2025]. Global and multigrid solvers obtain global coupling by assembling or coarsening a system each step, at a higher cost than VBD's local sweep. For example, on our beam, armadillo, and spot meshes, GPU multigrid PBD takes 5-8 seconds per substep at default parameters, two orders of magnitude above PAA's 50-90 ms per

time step. Within a strict real-time budget, the primary concern is the visual and physical quality a local solver can deliver, rather than the asymptotic convergence rate of an offline method. Because PAA inherits VBD's low per-iteration cost unchanged, a comprehensive cross-family benchmark involving rigorous, per-method tuning warrants its own dedicated study.

PAA is not universally beneficial. In collision-dominated scenarios, such as an object dropping onto a ground plane or being compressed between two rigid surfaces, the simulation error becomes high-frequency and spatially localized at the contact boundaries. Such localized residuals cannot be effectively accelerated by PAA's local mixing strategy. In an extended armadillo-drop test featuring ground and self-contact (360 substeps, 50 iterations per substep), the polynomial momentum inherent to both semi-iterative methods amplifies contact-induced numerical oscillations. While plain VBD remains stable throughout the entire sequence, VBD+Chebyshev and VBD+PAA diverge at substeps 86 and 66, respectively. Consequently, we recommend disabling acceleration entirely for contact-dominated scenes. On smooth, low-stiffness deformations, the iterate sequence is already well-behaved, and PAA's global correction brings small improvements (e.g., 1-2% at best). The period K admits per-scenario tuning: K=16 is robust across our beam bending and flatten recovery tests, while nose pull benefits from smaller K such as K=3. We recommend K=16 as a default.

PAA does not guarantee energy descent: the mixed iterate is a linear combination of past positions that may not lie in a descent direction. Across 1500 substeps per stiffness level, PAA produces higher per-step energy than its Chebyshev counterpart on 4-7% of high-stiffness substeps and up to 26% of low-stiffness substeps, but the cumulative gain on favorable substeps exceeds the cumulative loss by 3.6-46×. Adding a lightweight energy check to reject non-descent steps is straightforward. Across our test cases (beam bending at 30× and 100×, armadillo nose pull, and Case-3 flatten on five meshes), the guard rejected 0 of $\sim$ 27,200 AA firings, confirming the unguarded variant is empirically safe. Future work includes stress tests on large meshes (e.g., beyond 300K vertices) and extreme stretch scenarios. Further directions include porting PAA to other local solvers (e.g., XPBD, descent), composing it with a coarse multigrid or model-reduction correction, and an adaptive controller for K, m, and contact-aware disabling.

7 Conclusion

We show PAA composes with Chebyshev semi-iterative acceleration to accelerate Vertex Block Descent for implicit elastic body simulation. The key finding is that applying AA every K=16 iterations with a window of m=2 is both simpler and more effective than every-iteration AA, as the infrequent firing adds negligible overhead while the intervening Chebyshev steps improve the conditioning of AA's least-squares problem. Our method reduces elastic energy by 6-49% over the Chebyshev-only baseline across beam bending, nose pull, and flatten recovery tests on five mesh geometries, at under 5% wall-clock overhead on long simulations. The improvement is more significant on geometrically complex meshes (e.g., octopus: 49%) and extreme deformations (e.g., flatten recovery), and less significant on compact geometries with localized deformation (e.g., armadillo: 6%). Optionally combining with per-tet stiffness ramping [Giles et al. 2025] further improves convergence on high-stiffness scenarios. The PAA method requires no global matrix assembly, no mesh hierarchy, and adds a single integer parameter K to the existing VBD pipeline. Since PAA operates on the iterate sequence of any fixed-point solver, it is directly applicable to other local iterative methods (e.g., XPBD and Jacobi-preconditioned descent [Wang and Yang 2016]).

Acknowledgments

This work is in part supported by US NSF IIS-2005430 and the Qatar Research, Development and Innovation (QRDI) Council through the Academic Research Grant (ARG) award ARG01-0430-230047.

References

Steven S. An, Theodore Kim, and Doug L. James. 2008. Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. Graph.* 27, 5 (2008), 165. doi:10.1145/1409060.1409118

Donald G. M. Anderson. 1965. Iterative Procedures for Nonlinear Integral Equations. *J. ACM* 12, 4 (1965), 547–560. doi:10.1145/321296.321305

David Baraff and Andrew P. Witkin. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, Orlando, FL, USA, July 19-24, 1998*, Steve Cunningham, Walt Bransford, and Michael F. Cohen (Eds.). ACM, 43–54. doi:10.1145/280814.280821

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

Jan Bender, Matthias Müller, and Miles Macklin. 2017. A Survey on Position Based Dynamics. In *38th Annual Conference of the European Association for Computer Graphics, Eurographics 2017 - Tutorials, Lyon, France, April 24-28, 2017*, Adrien Bousseau and Diego Gutierrez (Eds.). Eurographics Association. doi:10.2312/EGT.20171034

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

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. Graph.* 33, 4 (2014), 154:1–154:11. doi:10.1145/2601097.2601116

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

Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popovic. 2002. Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph.* 21, 3 (2002), 586–593. doi:10.1145/566654.566622

Anka He Chen, Ziheng Liu, Yin Yang, and Cem Yuksel. 2024. Vertex Block Descent. *ACM Trans. Graph.* 43, 4 (2024), 116:1–116:16. doi:10.1145/3658179

Marco Fratarcangeli and Fabio Pellacini. 2015. Scalable Partitioning for Parallel Position Based Dynamics. *Comput. Graph. Forum* 34, 2 (2015), 405–413. doi:10.1111/CGF.12570

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.* 35, 6 (2016), 214:1–214:9. doi:10.1145/2980179.2982437

Theodore F. Gast, Craig A. Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M. Teran. 2015. Optimization Integrator for Large Time Steps. *IEEE Trans. Vis. Comput. Graph.* 21, 10 (2015), 1103–1115. doi:10.1109/TVCG.2015.2459687

Chris Giles, Elie Diaz, and Cem Yuksel. 2025. Augmented Vertex Block Descent. *ACM Trans. Graph.* 44, 4 (2025), 90:1–90:12. doi:10.1145/3731195

Eitan Grinspun, Petr Krysl, and Peter Schröder. 2002. CHARMS: a simple framework for adaptive simulation. *ACM Trans. Graph.* 21, 3 (2002), 281–290. doi:10.1145/566654.566578

Florian Hecht, Yeon Jin Lee, Jonathan Richard Shewchuk, and James F. O'Brien. 2012. Updated sparse cholesky factors for corotational elastodynamics. *ACM Trans. Graph.* 31, 5 (2012), 123:1–123:13. doi:10.1145/2231816.2231821

Thomas JR Hughes. 2003. *The finite element method: linear static and dynamic finite element analysis.* Courier Corporation.

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 (2025), 44:1–44:15. doi:10.1145/3731183

Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022. Penetration-free projective dynamics on the GPU. *ACM Trans. Graph.* 41, 4 (2022), 69:1–69:16. doi:10.1145/3528223.3530069

Chunlei Li, Peng Yu, Tiantian Liu, Siyuan Yu, Yuting Xiao, Shuai Li, Aimin Hao, Yang Gao, and Qinping Zhao. 2025. MGPBD: A Multigrid Accelerated Global XPBD Solver. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*. 1–11.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Trans. Graph.* 36, 3 (2017), 23:1–23:16. doi:10.1145/2990496

Jia-Ming Lu, Tailing Yuan, Zhe-Han Mo, and Shi-Min Hu. 2025. Fast Galerkin Multigrid Method for Unstructured Meshes. *ACM Trans. Graph.* 44, 6 (2025), 179:1–179:16. doi:10.1145/3763327

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games, MIG 2016, Burlingame, California, USA, October 10-12, 2016*, Michael Neff, Roland Geraerts, and Hubert P. H. Shum (Eds.). ACM, 49–54. doi:10.1145/2994258.2994272

Miles Macklin, Kier Storey, Michelle Lu, Pierre Terdiman, Nuttapong Chentanez, Stefan Jeschke, and Matthias Müller. 2019. Small steps in physics simulation. In *Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2019, Los Angeles, CA, USA, July 26-28, 2019*, Sung-Hee Lee, Craig A. Schroeder, Stephen N. Spencer, Christopher Batty, and Jin Huang (Eds.). ACM, 2:1–2:7. doi:10.1145/3309486.3340247

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus H. Gross. 2011. Example-based elastic materials. *ACM Trans. Graph.* 30, 4 (2011), 72. doi:10.1145/2010324.1964967

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, San Antonio, TX, USA, July 21-22, 2002*, Jessica K. Hodgins, Michiel van de Panne, Michael F. Cohen, and Nancy S. Pollard (Eds.). ACM, 49–54. doi:10.1145/545261.545269

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *J. Vis. Commun. Image Represent.* 18, 2 (2007), 109–118. doi:10.1016/J.JVCIR.2007.01.005

Yue Peng, Bailin Deng, Juyong Zhang, Fanyu Geng, Wenjie Qin, and Ligang Liu. 2018. Anderson acceleration for geometry optimization and physics simulation. *ACM Trans. Graph.* 37, 4 (2018), 42. doi:10.1145/3197517.3201290

Breannan Smith, Fernando de Goes, and Theodore Kim. 2018. Stable Neo-Hookean Flesh Simulation. *ACM Trans. Graph.* 37, 2 (2018), 12. doi:10.1145/3180491

Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph.* 34, 6 (2015), 245:1–245:13. doi:10.1145/2816795.2818081

Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ronald Fedkiw. 2005. Robust quasistatic finite elements and flesh simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2005, Los Angeles, CA, USA, July 29-31, 2005*, Demetri Terzopoulos, Victor B. Zordan, Ken Anjyo, and Petros Faloutsos (Eds.). ACM, 181–190. doi:10.2312/SCA/SCA05/181-190

Quoc-Minh Ton-That, Paul G. Kry, and Sheldon Andrews. 2023. Parallel block Neo-Hookean XPBD using graph clustering. *Comput. Graph.* 110 (2023), 1–10. doi:10.1016/J.CAG.2022.10.009

Homer F. Walker and Peng Ni. 2011. Anderson Acceleration for Fixed-Point Iterations. *SIAM J. Numer. Anal.* 49, 4 (2011), 1715–1735. doi:10.1137/10078356X

Huamin Wang. 2015. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph.* 34, 6 (2015), 246:1–246:9. doi:10.1145/2816795.2818063

Huamin Wang and Yin Yang. 2016. Descent methods for elastic body simulation on the GPU. *ACM Trans. Graph.* 35, 6 (2016), 212:1–212:10. doi:10.1145/2980179.2980236

Botao Wu, Zhendong Wang, and Huamin Wang. 2022. A GPU-based multilevel additive schwarz preconditioner for cloth and deformable body simulation. *ACM Trans. Graph.* 41, 4 (2022), 63:1–63:14. doi:10.1145/3528223.3530085

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.* 29, 2 (2010), 16:1–16:18. doi:10.1145/1731047.1731054