Fast Galerkin Multigrid Method for Unstructured Meshes

JIA-MING LU, Tsinghua University, China
TAILING YUAN, Independent Researcher, China
ZHE-HAN MO, Tsinghua University, China
SHI-MIN HU*, Tsinghua University, China

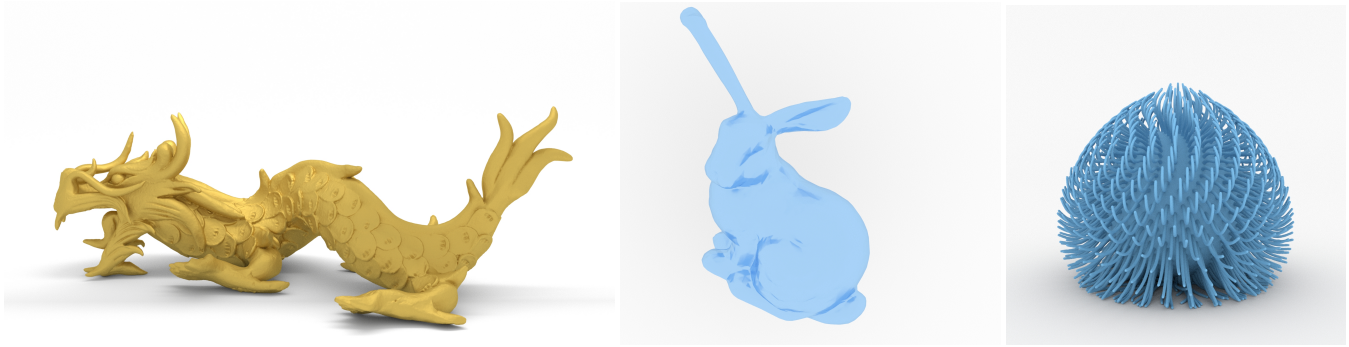


Fig. 1. Our method efficiently handles high-resolution elastodynamic simulations with complex contact mechanics. Left: A million-point Stanford dragon dropping and deforming. Middle: Detailed cloth-like draping of a million-point bunny model. Right: A squishy ball demonstrating intricate self-contact and large deformations.

We present a novel multigrid solver framework that significantly advances the efficiency of physical simulation for unstructured meshes. While multigrid methods theoretically offer linear scaling, their practical implementation for deformable body simulations faces substantial challenges, particularly on GPUs. Our framework achieves up to 6.9× speedup over traditional methods through an innovative combination of matrix-free vertex block Jacobi smoothing with a Full Approximation Scheme (FAS), enabling both piecewise constant and linear Galerkin formulations without the computational burden of dense coarse matrices. Our approach demonstrates superior performance across varying mesh resolutions and material stiffness values, maintaining consistent convergence even under extreme deformations and challenging initial configurations. Comprehensive evaluations against state-of-the-art methods confirm our approach achieves lower simulation error with reduced computational cost, enabling simulation of tetrahedral meshes with over one million vertices at approximately one frame per second on modern GPUs.

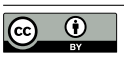
CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: physics-based animation

ACM Reference Format:
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, Article 179 (December 2025), 16 pages. https://doi.org/10.1145/3763327

*Corresponding author

Authors' Contact Information: Jia-Ming Lu, Tsinghua University, Beijing, China, jaimeyzzz@outlook.com; Tailing Yuan, Independent Researcher, Beijing, China, yuantailing@gmail.com; Zhe-Han Mo, Tsinghua University, Beijing, China, mozh23@mails.tsinghua.edu.cn; Shi-Min Hu, Tsinghua University, Beijing, China, shimin@tsinghua.edu.cn.



ACM 1557-7368/2025/12-ART179
https://doi.org/10.1145/3763327

1 Introduction

Physical simulation is fundamental to computer graphics, driving applications from visual effects to interactive experiences. As modern applications demand increasingly complex, high-fidelity simulations, computational efficiency has become a critical challenge, particularly for real-time and large-scale production environments.

The core challenge lies in solving nonlinear systems describing physical dynamics, and can be written to a nonlinear optimization form with

$$\boldsymbol{x} = \arg\min_{\boldsymbol{x}} \frac{1}{2\Delta t^2}(\boldsymbol{x}-\hat{\boldsymbol{x}})^\top \boldsymbol{M}(\boldsymbol{x}-\hat{\boldsymbol{x}}) + E(\boldsymbol{x}), \tag{1}$$

where $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{x}$ the current state, $\hat{\boldsymbol{x}}$ the predicted state, Δt the time step and $E(\boldsymbol{x})$ the potential energy. While Newton's method offers quadratic convergence, its practical application faces significant challenges, particularly in solving the linear system at each iteration. High condition numbers—common in ill-conditioned or high-DOF systems—make the linear solve numerically challenging. This issue compounds with system size, as both per-iteration cost and iteration count grow. Traditional solvers often struggle to maintain efficiency across varying simulation scenarios, especially with complex materials or real-time constraints, creating a need for more robust and scalable solutions.

Multigrid methods represent a powerful approach to achieving linear computational scaling with respect to system degrees of freedom, showing particular success in regular grid structures such as fluid simulations. Among these, the Galerkin method is a highly effective approach that automatically generates coarse-level operators through projection without requiring explicit rediscretization. This approach maintains level consistency through variational principles, ensuring proper energy minimization across scales. The Galerkin method preserves important mathematical properties of the original

system such as symmetry and positive-definiteness, while naturally adapting to problems with complex geometries and heterogeneous material properties. It also provides theoretically optimal convergence rates independent of problem size and handles boundary conditions consistently across all levels.

The extension of Galerkin multigrid methods to unstructured meshes and dynamically deforming systems presents significant challenges. In volumetric elastodynamics, where the domain is typically discretized using tetrahedral elements, traditional multigrid approaches face several limitations. The irregular mesh structure complicates the construction of coherent coarse levels, and when smooth linear interpolation weights are used, the coarse matrix loses its sparsity property. The resulting dense coarse matrices become computationally intractable both in terms of storage and computation. These challenges are particularly pronounced in high-resolution simulations where multigrid methods would potentially offer the greatest benefits.

Modern GPU architectures, while offering tremendous computational power, present additional challenges for multigrid method implementation. The hierarchical nature of multigrid algorithms suggests sequential execution, making optimal GPU utilization non-trivial. Key challenges include managing irregular data access patterns, balancing loads across different grid levels, and minimizing synchronization overhead between GPU kernels. These efficiency considerations become particularly critical for real-time applications such as interactive virtual surgery or game physics. The complex interaction between mesh topology, numerical properties, and GPU architecture characteristics demands careful algorithm design to achieve optimal performance while maintaining numerical robustness.

Prior research has explored some solutions, such as piecewise constant weights [Wu et al. 2022; Xian et al. 2019] to avoid dense coarse matrices, and multilevel additive Schwarz methods with domain decomposition for parallel computation of multilevel preconditioners [Wu et al. 2022]. However, these approaches face several limitations: they require system matrix assembly and linear system solving, incurring substantial computational costs for matrix evaluation and precomputation. Moreover, they struggle with heterogeneous materials and cannot leverage smoother interpolation methods like piecewise linear operators to improve convergence rates.

Our work addresses these fundamental challenges through a novel approach that combines theoretical rigor with practical efficiency. The key contributions of our research are:

- A highly efficient matrix-free smoothing operator optimized for parallel architectures. This smoother outperforms both previous methods and standard solvers, achieving exceptional efficiency even as a standalone solver. Our implementation optimally balances computational throughput with memory bandwidth utilization, resulting in significant performance improvements.
- A specialized FAS multigrid method integrated with our novel smoother for nonlinear equation systems. This combination delivers faster convergence rates than existing approaches while maintaining robust numerical behavior across diverse simulation scenarios.
- An optimized piecewise constant Galerkin formulation for the FAS scheme that reduces computational overhead on coarser grids by strategically eliminating interior element calculations. This approach preserves accuracy while substantially improving performance.
- A versatile Galerkin formulation framework for unstructured meshes, offering both piecewise constant and linear variants without the prohibitive computational costs typically associated with dense matrices on coarse levels.

2 Related Work

Physics-based simulation in computer graphics predominantly involves solving nonlinear systems. The implicit integration framework, pioneered by [Baraff and Witkin 1998], has established itself as the standard approach for stable simulation, particularly excelling in handling stiff systems and large time steps. While extensive research has been conducted in this field over the past decades, we focus our discussion on works most relevant to our method.

The field has evolved significantly through optimization-based approaches. Notable contributions from [Liu et al. 2013] and [Bouaziz et al. 2014] advanced the application of optimization techniques in deformable body simulation by introducing efficient local-global solving strategies, establishing new paradigms for physics-based simulation. [Gast et al. 2015] further developed these concepts by creating optimization formulations that maintain stability even with large time steps, significantly improving simulation robustness. More recently, [Trusty et al. 2022] introduced a mixed variational framework for implicit time integration of elastica that bridges the gap between accuracy and efficiency. Their approach employs sequential quadratic programming to combine the precision typically associated with Newton solvers with the computational efficiency of fast simulation methods.

The advent of GPU computing has sparked innovative solutions for parallel computation. [Wang and Yang 2016] introduced a Jacobi-preconditioned gradient descent method specifically designed to harness parallel computing capabilities. [Wang 2015] enhanced convergence rates through Chebyshev acceleration while preserving implementation simplicity. In recent developments, [Chen et al. 2024] achieved substantial performance improvements through a vertex block descent method that employs vertex coloring for parallel Gauss-Seidel iterations on GPUs. While their approach offers an elegant solution for GPU-based Gauss-Seidel iterations, block Jacobi methods have established a strong theoretical foundation for parallel preconditioning [Hegland and Saylor 1992; Sicot et al. 2008]. These methods naturally align with parallel architectures and avoid the synchronization overhead inherent in colored schemes. Our proposed smoother builds upon this tradition of block Jacobi preconditioning, offering comparable convergence properties with superior parallelization characteristics.

Despite these advances, physics-based simulation continues to face significant challenges. Key areas requiring further research include accelerating simulation rates for systems with high degrees of freedom, expanding the range of supported material properties (especially for highly stiff materials), and developing algorithms

that fully utilize the computational capabilities of modern GPU architectures.

Multigrid Methods in Graphics. Multigrid methods have demonstrated remarkable success in achieving linear scaling with system size across various simulation domains.

Multigrid methods can be used to solve linear systems on unstructured curved surfaces efficiently in [Liu et al. 2021]. In fluid simulation [Wang et al. 2024], particularly for Eulerian frameworks where grids are regular, multigrid methods have shown exceptional performance. While the problem domain differs significantly from our focus, notable contributions include [Aanjaneya et al. 2017; Chentanez and Müller 2011; Dick et al. 2015; Ferstl et al. 2014; McAdams et al. 2010; Shao et al. 2022; Weber et al. 2015]. For Material Point Method simulations, [Wang et al. 2020] developed a specialized hierarchical optimization algorithm combining Galerkin multigrid with quasi-Newton methods, achieving robust and parameter-free implicit time stepping. Cloth simulation represents another area where multigrid methods have been extensively explored. [Lee et al. 2010] introduced a multi-resolution approach that identifies regions with smooth solutions to solve linear systems in a reduced space. [Jeon et al. 2013] developed a novel soft-constraint formulation for multigrid solvers that effectively propagates point and sliding constraints across mesh levels. [Tamstorf et al. 2015] advanced the field by applying smooth aggregation multigrid methods to cloth simulation. [Wang et al. 2018] proposed a parallel-friendly full multigrid method for cloth simulation, while [Wu et al. 2022] focused on a multi-level additive Schwarz preconditioner with domain decomposition, specifically designed for GPU acceleration. More recently, [Ruan et al. 2024] introduced MiNNIE, a comprehensive framework for real-time simulation of nonlinear near-incompressible elastics that incorporates a mixed FEM formulation with pressure stabilization and features a GPU multigrid solver with vertex Vanka smoother. For practical implementations, [Naumov et al. 2015] developed AMGX, a GPU-accelerated algebraic multigrid library that provides efficient parallel implementations of hierarchical solvers.

Our method relates to elastic body simulation using multigrid approaches. Early contributions include [Georgii and Westermann 2006], which introduced a multigrid solver capable of handling linear, corotational, and nonlinear Green strain models. [Zhu et al. 2010] developed a parallel multigrid framework for high-resolution elastic simulation that efficiently manages complex geometry and boundary conditions using structured regular grids. This principle was later adapted by [McAdams et al. 2011] for high-performance soft tissue simulation, combining vectorized SVD, efficient multigrid solving, and collision handling to achieve near-interactive rates for production-quality character skinning. However, these approaches face significant challenges when applied to unstructured meshes. The Scalable Galerkin Method (SGM) [Xian et al. 2019] employs a Galerkin approach with piecewise constant weights in skinning space coordinates, using geometric construction for hierarchy layers. While innovative, this method has two main limitations: its sequential computation across layers prevents full GPU utilization for coarse layers, and it is restricted to homogeneous materials. The more recent Multilevel Additive Schwarz (MAS) method [Wu et al. 2022], while primarily focused on cloth simulation, also demonstrates results for elastic bodies. It employs domain decomposition as a preconditioner with parallel computation across layers. However, this approach faces several constraints: it functions only as a preconditioner rather than a standalone solver, incurs substantial computational costs for matrix evaluation and precomputation, and has been validated only for systems up to approximately 500K points. Additionally, like SGM, it struggles with heterogeneous materials. Both methods utilize piecewise constant weights or selection matrices for coarse projection, as this approach preserves sparsity in coarse layer matrices and avoids the prohibitive storage and computational costs associated with dense matrices.

Beyond traditional multigrid methods, various hierarchical approaches have been explored. [Müller 2015] applied hierarchical structures to constraint solving, explicitly constructing coarse stretch constraints for cloth simulation to reduce unrealistic stretching. Recently, [Mercier-Aubin and Kry 2024] developed a multi-layer solver for Position Based Dynamics (PBD) that leverages adaptive rigidity within a hierarchical framework to accelerate constraint resolution. Progressive simulation methods have also adopted hierarchical structures, as demonstrated by [Zhang et al. 2022] and [Zhang et al. 2023], which incrementally increase simulation detail. This approach was further extended in [Zhang et al. 2024], combining spatial hierarchies with time stepping through diagonal refinement.

Full Approximation Scheme. The Full Approximation Scheme, introduced by [Brandt 1977], is a nonlinear multigrid method that addresses nonlinear problems directly across multiple resolutions without requiring linearization. Unlike traditional linear multigrid methods that only transfer correction terms between levels, FAS transfers both the solution and the residual, making it particularly effective for nonlinear elasticity problems. In computer graphics, FAS has demonstrated its utility through various applications. [Otaduy et al. 2007] employed it to incorporate collision handling at coarse resolutions in adaptive deformation, while [McAdams et al. 2011] utilized it for efficient high-resolution elastic simulation. The method's key strength lies in its preservation of nonlinear characteristics across all resolution levels, enabling robust convergence for highly nonlinear elastodynamic problems. Our work leverages FAS to construct a simple yet efficient method suitable for fast, parallel GPU computation.

3 Method

Having presented a comprehensive review of multigrid methods, we note that Galerkin multigrid methods are widely adopted for their natural handling of boundary conditions through full-space problem construction with restriction and prolongation operators. However, when applied to unstructured meshes, these methods face a significant limitation: they generate dense coarse matrices for irregular grids, leading to computational inefficiency. Previous work, such as [Xian et al. 2019], addressed this by implementing piecewise-constant weights to mitigate element explosion in coarse problems.

While piecewise-linear operators could potentially offer smoother interpolation and better convergence for multigrid methods, their naive implementation results in dense coarse matrices, making them

impractical for unstructured meshes. However, upon reviewing this approach, we discovered a potential solution. Our key observation stems from understanding the role of relaxation in multigrid methods: it primarily addresses high-frequency errors within each layer, where local computations are sufficient. This is exemplified by methods like Jacobi-conditioned gradient descent [Wang and Yang 2016], which effectively operates using local information. This locality principle suggests that full matrix construction and global information processing may be unnecessary for high-frequency error reduction.

To formalize our method, we first review the classical multigrid framework for linear systems. Consider a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. For a two-layer multigrid V-cycle with prolongation matrix $\boldsymbol{P}$ and restriction matrix $\boldsymbol{R} = \boldsymbol{P}^\top$, the algorithm proceeds as follows:

ALGORITHM 1: A two-grid v-cycle for solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

1 Initialize $\boldsymbol{x} = \boldsymbol{x}_0$;
2 Apply pre-smoothing iterations to update $\boldsymbol{x}$;
3 Compute residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$;
4 Restrict residual $\boldsymbol{r}_c = \boldsymbol{P}^\top \boldsymbol{r}$;
5 Solve coarse system $\boldsymbol{A}_c \boldsymbol{e}_c = \boldsymbol{r}_c$;
6 Update solution $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{P}\boldsymbol{e}_c$;
7 Apply post-smoothing to $\boldsymbol{x}$;

The Full Approximation Scheme to nonlinear problems follows a similar structure but with crucial modifications:

ALGORITHM 2: A two-grid FAS v-cycle for solving $\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{b}$

1 Initialize $\boldsymbol{x} = \boldsymbol{x}_0$;
2 Apply pre-smoothing iterations to update $\boldsymbol{x}$;
3 Compute residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}(\boldsymbol{x})$;
4 Compute restricted residual $\boldsymbol{r}_c = \boldsymbol{P}^\top \boldsymbol{r}$ and solution approximation $\overline{\boldsymbol{x}}_c = \boldsymbol{P}^\top \boldsymbol{x}$;
5 Solve nonlinear coarse system $\boldsymbol{A}_c(\boldsymbol{x}_c) = \boldsymbol{A}_c(\overline{\boldsymbol{x}}_c) + \boldsymbol{r}_c$;
6 Update solution $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{P}(\boldsymbol{x}_c - \overline{\boldsymbol{x}}_c)$;
7 Apply post-smoothing to $\boldsymbol{x}$;

The distinguishing feature of FAS lies in its nonlinear coarse-grid problem and the transfer of the full solution rather than just the error correction between levels.

We now examine the Galerkin projection in both linear and nonlinear contexts. For linear systems, the Galerkin projection constructs the coarse matrix as $\boldsymbol{A}_c = \boldsymbol{P}^\top \boldsymbol{A}\boldsymbol{P}$. This naturally extends to nonlinear problems as $\boldsymbol{A}_c(\boldsymbol{x}_c) = \boldsymbol{P}^\top \boldsymbol{A}(\boldsymbol{P}\boldsymbol{x}_c)$. The coarse-grid equation can thus be reformulated as:

$$\boldsymbol{g}(\boldsymbol{x}_c) = \boldsymbol{P}^\top \boldsymbol{A}(\boldsymbol{P}\boldsymbol{x}_c) - \boldsymbol{P}^\top \boldsymbol{A}(\boldsymbol{P}\overline{\boldsymbol{x}}_c) - \boldsymbol{r}_c = 0. \tag{2}$$

To solve this nonlinear equation utilizing only local information, we employ Jacobi-preconditioned gradient descent, which efficiently reduces high-frequency errors while requiring only diagonal matrix elements. For a nonlinear function $g(x)$, the update takes the form:

$$\Delta \boldsymbol{x}_c = -\mathrm{diag}^{-1}(\boldsymbol{g}'(\boldsymbol{x}_c))\boldsymbol{g}(\boldsymbol{x}_c). \tag{3}$$

With initial guess $\boldsymbol{x}_c = \overline{\boldsymbol{x}}_c$, we obtain:

$$\begin{aligned} \boldsymbol{g}(\boldsymbol{x}_c) &= \boldsymbol{P}^\top \boldsymbol{A}(\boldsymbol{P}\overline{\boldsymbol{x}}_c) - \boldsymbol{P}^\top \boldsymbol{A}(\boldsymbol{P}\overline{\boldsymbol{x}}_c) - \boldsymbol{r}_c \\ &= -\boldsymbol{r}_c \\ &= -\boldsymbol{P}^\top(\boldsymbol{b} - \boldsymbol{A}(\boldsymbol{x})). \end{aligned} \tag{4}$$

The preconditioner takes the form:

$$\begin{aligned} \mathrm{diag}(\boldsymbol{g}'(\boldsymbol{x}_c)) &= \mathrm{diag}(\boldsymbol{P}^\top \boldsymbol{A}'(\boldsymbol{P}\overline{\boldsymbol{x}}_c)\boldsymbol{P}) \\ &= \mathrm{diag}(\boldsymbol{P}^\top \boldsymbol{A}'(\boldsymbol{P}\boldsymbol{P}^\top \boldsymbol{x})\boldsymbol{P}). \end{aligned} \tag{5}$$

Combining these expressions in Equation (3) yields:

$$\Delta \boldsymbol{x}_c = \mathrm{diag}^{-1}(\boldsymbol{P}^\top \boldsymbol{A}'(\boldsymbol{P}\boldsymbol{P}^\top \boldsymbol{x})\boldsymbol{P})\boldsymbol{P}^\top(\boldsymbol{b} - \boldsymbol{A}(\boldsymbol{x})). \tag{6}$$

A key insight emerges: explicit computation of $\boldsymbol{x}_c$ is unnecessary; we only need the prolongation operator for update calculations. We observe that replacing the projection-interpolation term $\boldsymbol{P}\boldsymbol{P}^\top \boldsymbol{x}$ with $\boldsymbol{x}$ not only simplifies the implementation but also improves convergence by preserving fine-scale information. This modification, while affecting the preconditioner, maintains solution accuracy. The resulting algorithm for solving Equation (1) is presented in Algorithm 3. For notational clarity, let $\boldsymbol{H}(\boldsymbol{x}) = \nabla^2 \phi(\boldsymbol{x})$ where $\phi(\boldsymbol{x})$ is the objective function in Equation (1).

ALGORITHM 3: A two-grid FAS v-cycle for $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} \phi(\boldsymbol{x})$

1 Initial guess $\boldsymbol{x} = \boldsymbol{x}_0$;
2 Pre-smooth, update $\boldsymbol{x} \leftarrow \boldsymbol{x} - (\mathrm{diag}(\boldsymbol{H}(\boldsymbol{x})))^{-1}\nabla\phi(\boldsymbol{x})$;
3 Solve the coarse problem and get the coarse updation $\Delta \boldsymbol{x}_c = -(\mathrm{diag}(\boldsymbol{P}^\top \boldsymbol{H}(\boldsymbol{x})\boldsymbol{P}))^{-1}\boldsymbol{P}^\top(\nabla\phi(\boldsymbol{x}))$;
4 Update $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{P}\Delta\boldsymbol{x}_c$;
5 Post-smooth, update $\boldsymbol{x} \leftarrow \boldsymbol{x} - (\mathrm{diag}(\boldsymbol{H}(\boldsymbol{x})))^{-1}\nabla\phi(\boldsymbol{x})$;

Our algorithm offers significant computational advantages over the standard two-layer V-cycle multigrid method. The primary computations reduce to: the diagonal elements $\boldsymbol{D}_c = \mathrm{diag}(\boldsymbol{P}^\top \boldsymbol{H}(\boldsymbol{x})\boldsymbol{P})$ and the function residual projection $\boldsymbol{P}^\top \nabla\phi(\boldsymbol{x})$. The latter, being a vector projection, is naturally amenable to parallel implementation. The computational complexity primarily lies in evaluating $\boldsymbol{D}_c$, which takes the form:

$$(\boldsymbol{D}_c)_{ij} = \begin{cases} \boldsymbol{p}_i^\top \boldsymbol{H}(\boldsymbol{x})\boldsymbol{p}_i, & i = j \\ 0, & i \neq j \end{cases} \tag{7}$$

For each nonzero diagonal element $(\boldsymbol{D}_c)_{ii}$, computation requires collecting matrix elements of $\boldsymbol{H}(\boldsymbol{x})$ associated with coarse element i, forming a block $\boldsymbol{B}(\boldsymbol{x}, i)$. Given the i-th row of $\boldsymbol{P}$ denoted as $\boldsymbol{p}_i$, we compute $(\boldsymbol{D}_c)_{ii} = \boldsymbol{p}_i^\top \boldsymbol{B}(\boldsymbol{x}, i)\boldsymbol{p}_i$. This formulation eliminates the need to construct and store dense coarse-level matrices. The efficient implementation of this approach is detailed in the following sections.

4 Fast Matrix-Free Smoother

4.1 Choice of Smoother

Block Size Considerations. The choice of block size in multigrid smoothers involves a fundamental trade-off: larger blocks achieve better convergence per iteration through comprehensive local information exchange but require more computational resources, while

smaller blocks offer lower per-iteration cost but may need more iterations. Modern GPU architectures add complexity to this trade-off, as different block sizes align with different levels of GPU hierarchy, making block size selection both a mathematical and hardware-aware design choice.

A key question is whether we can adopt larger GPU-friendly blocks similar to MAS [Wu et al. 2022]. However, our approach differs fundamentally from MAS in design philosophy. While MAS focuses primarily on accelerating iterative convergence, our method aims to leverage both faster convergence and the advantages of frequent updates with small time steps. This design difference leads to distinct update frequency requirements: MAS operates as a preconditioner with relatively static matrix information, while our method necessitates frequent matrix updates at each level.

Constructing larger blocks at coarse levels would create significant overhead, as each coarse node corresponds to many fine-level nodes, making the assembly process substantially more expensive than fine-level operations. Our goal is to balance smoother iteration efficiency with frequent update overhead, ensuring multigrid benefits are not overshadowed by assembly costs.

Our Approach. Base on these considerations, we focus on vertex-level block selection and demonstrate its advantages in achieving this balance. We employ a Jacobi gradient descent smoother with vertex-based blocking inspired by VBD [Chen et al. 2024]. While standard Jacobi methods utilize scalar diagonal elements, our vertex block approach processes 3×3 blocks corresponding to vertex degrees of freedom, delivering superior convergence with comparable computational overhead.

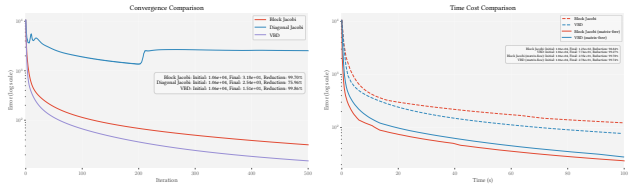


Fig. 2. Smoother performance comparison on a 100K-element bunny drape simulation. Left: convergence comparison between diagonal Jacobi, vertex block Jacobi, and VBD methods. Right: time-to-error comparison between different smoother implementations. Our matrix-free vertex block Jacobi method achieves the fastest error reduction, outperforming all alternatives.

We evaluated smoother performance using a 100K-element bunny drape simulation with 1m/s initial velocity, Young's modulus of 10^9 Pa, and fixed step size ($\alpha = 0.7$). As shown in Figure 2, our vertex block Jacobi method demonstrates superior performance in both convergence rate and computational efficiency.

The left plot in Figure 2 shows that our matrix-free vertex block Jacobi method achieves faster time-to-solution performance than matrix-free VBD, despite VBD's superior per-iteration convergence rate. This advantage stems from our approach requiring significantly less computation time per iteration, providing an optimal balance between mathematical effectiveness and computational efficiency.

The right plot in Figure 2 compares matrix-explicit and matrix-free variants for both block Jacobi and VBD methods (implementation details in Section 4.2). Our matrix-free implementations demonstrate more than an order of magnitude speedup compared to their matrix-explicit counterparts, highlighting the substantial performance advantages of our matrix-free formulation across different smoother types.

4.2 Matrix-Free Vertex Block Jacobi

In Galerkin methods, fine points connect to multiple coarse parents, creating overlapping blocks. While computationally lighter than full coarse matrix assembly, computing diagonal elements $(\boldsymbol{D}_c)_{ii} = \boldsymbol{p}_i^\top \boldsymbol{B}(\boldsymbol{x}, i)\boldsymbol{p}_i$ remains expensive. This cost is primarily due to elastic terms where the tet-to-vertex ratio typically ranges from 5-8x.

Previous work [Chen et al. 2024] proposes a GPU warp-level gather strategy where each point aggregates contributions from neighboring elements. This approach enables efficient point coloring with minimal color counts and leverages GPU warp-level synchronization. However, it induces 4x memory reads as each element is processed four times. To reduce memory overhead, we adopt an element-wise scatter approach. However, the naive implementation requires computing 12×12 matrix per element and updating 3×3 blocks, this proves GPU-inefficient due to thread-level resource constraints and memory bandwidth limitations when handling 144-element matrices.

To overcome the computational bottlenecks, we present an efficient matrix-free formulation for computing $(\boldsymbol{D}_c)_{ii}$. Our approach builds upon the Hessian formulation of the stable neo-Hookean model [Kim and Eberle 2020]:

$$\frac{\partial^2 \Psi}{\partial \boldsymbol{x}^2} = \left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\right)^\top \frac{\partial^2 \Psi}{\partial \boldsymbol{F}^2} \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}, \tag{8}$$

where Ψ denotes the strain energy density function, $\boldsymbol{F}$ represents the deformation gradient, and $\boldsymbol{x}$ contains the nodal positions of the tetrahedral mesh. The key computation involves the diagonal contribution from a fine element to its associated coarse parent i:

$$(\boldsymbol{D}_c)_i = \boldsymbol{p}_i^\top \frac{\partial^2 \Psi}{\partial \boldsymbol{x}^2} \boldsymbol{p}_i = \boldsymbol{p}_i^\top \left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\right)^\top \frac{\partial^2 \Psi}{\partial \boldsymbol{F}^2} \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} \boldsymbol{p}_i. \tag{9}$$

The prolongation operator $\boldsymbol{p}_i \in \mathbb{R}^{12\times 3}$ takes the form:

$$\begin{pmatrix} w_0 \boldsymbol{I}_3 \\ w_1 \boldsymbol{I}_3 \\ w_2 \boldsymbol{I}_3 \\ w_3 \boldsymbol{I}_3 \end{pmatrix}. \tag{10}$$

Here, $\boldsymbol{I}_3$ represents the 3×3 identity matrix, and the scalars $w_k \in \mathbb{R}$ ($k = 0, 1, 2, 3$) encode the interpolation weights between the vertices of a fine tetrahedron and a coarse vertex. These weights are distinct from the barycentric weights used in standard prolongation operations—w_k quantifies the interpolation relationship between coarse vertex i and the k-th vertex of the fine tetrahedron, vanishing ($w_k = 0$) when no connection exists between the vertices.

Following [Kim and Eberle 2020], $\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}$ is a 9×12 matrix of $a\boldsymbol{I}_3$ blocks, where a corresponds to elements of matrix $\boldsymbol{B}$. The product

of $\frac{\partial F}{\partial x}$ and p_i yields:

$$U = \frac{\partial F}{\partial x} p_i = \begin{pmatrix} u_0 I_3 \\ u_1 I_3 \\ u_2 I_3 \end{pmatrix}, \tag{11}$$

with coefficients:

$$\begin{aligned} u_0 &= (w_1 - w_0)B_{00} + (w_2 - w_0)B_{10} + (w_3 - w_0)B_{20} \\ u_1 &= (w_1 - w_0)B_{01} + (w_2 - w_0)B_{11} + (w_3 - w_0)B_{21} \\ u_2 &= (w_1 - w_0)B_{02} + (w_2 - w_0)B_{12} + (w_3 - w_0)B_{22}. \end{aligned} \tag{12}$$

For the stable neo-Hookean model, $\frac{\partial^2 \Psi}{\partial F^2}$ consists of:

$$\frac{\partial^2 \Psi}{\partial F^2} = \mu I_9 + (\lambda(J-1) - \mu) H_{\text{volume}} + \lambda \hat{g} \hat{g}^\top, \tag{13}$$

where μ and λ are Lamé parameters, J is the determinant of F, $\hat{g}$ is the flattened vector of $\frac{\partial J}{\partial F}$, and H_{volume} is constructed from cross matrices of F's columns f_0, f_1, f_2. We can get a vector z by:

$$z = \hat{g}^\top U = u_0(f_1 \times f_2) + u_1(f_2 \times f_0) + u_2(f_0 \times f_1). \tag{14}$$

Substituting these terms into Equation (9) yields an simplified element contribution:

$$(D_c)_i = p_i^\top \frac{\partial^2 \Psi}{\partial x^2} p_i = \mu(u_0^2 + u_1^2 + u_2^2) I_3 + \lambda z z^\top. \tag{15}$$

A notable observation is that the H_{volume} contribution vanishes due to its specialized structure as a cross matrix of cross matrices. The gradient computation follows a similar simplification, here vec is a function which flatten a matrix:

$$\begin{aligned} (b_c)_i &= p_i^\top \frac{\partial \Psi}{\partial x} = p_i^\top \left(\frac{\partial F}{\partial x}\right)^\top \text{vec}\left(\frac{\partial \Psi}{\partial F}\right) \\ &= \mu(u_0 f_0 + u_1 f_1 + u_2 f_2) + (\lambda(J-1) - \mu) z. \end{aligned} \tag{16}$$

This matrix-free formulation dramatically reduces computational overhead by replacing the traditional 12×12 matrix operations with efficient 3×1 vector operations per tetrahedral element. With precomputed u_0, u_1, u_2, the runtime complexity reduces to computing the deformation gradient F followed by a series of lightweight vector operations. Each element requires this computation for its associated coarse elements—typically four to five times per element, corresponding to cases where a fine element either resides within a coarse element (four times) or intersects a face between two elements (five times). While we derived this formulation for the stable neo-Hookean model, the matrix-free approach generalizes to other constitutive models, offering similar computational benefits.

5 Multigrid Method Implementation

Our method employs geometric approaches for hierarchy construction and Galerkin operator computation. We present two distinct formulations: piece-wise constant and piece-wise linear, each with unique characteristics and performance profiles. Our comprehensive evaluations demonstrate that the piece-wise constant Galerkin method delivers superior GPU performance due to significantly reduced computational requirements in coarse layers. While we present both approaches for completeness, readers should note that our constant Galerkin implementation represents the recommended configuration for practical applications. The linear Galerkin method is included as an alternative formulation to illustrate the design space, though it does not achieve the same computational efficiency in our GPU-oriented implementation.

5.1 Hierarchy Construction

Our hierarchy construction leverages the algorithmic framework established in Section 3, implementing two distinct approaches for different computational trade-offs.

5.1.1 Piece-wise Constant Approach. We formulate the hierarchy construction as a graph clustering problem, establishing non-overlapping tetrahedral partitions with a piece-to-one mapping between fine points and coarse parents. Unlike classical multigrid algorithms that construct hierarchies sequentially, our method builds different layers independently, which we found yields optimal convergence.

Specifically, we adopt the strategy from [Mahmoud et al. 2021] to construct equally-sized tetrahedral pieces, employing a coarsening ratio of 8 to scale piece sizes between levels. This approach aligns with prior works such as SGM and MAS, which employ similar piece-wise constant formulations through their respective "piece-wise constant matrix" and "selection matrix."

A key optimization enabled by this coarse structure construction is that during subsequent multigrid operations, coarse layers need not process all tetrahedra, as internal tetrahedra do not influence the final result when applying coarsening operators. As illustrated in Figure 3, only boundary elements subject to external forces need to be processed during restriction and prolongation operations, which significantly reduces computational overhead in coarse layers.

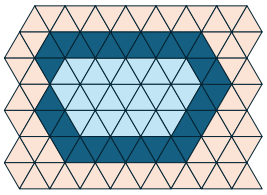


Fig. 3. **Illustration of piece-wise constant discretization on a triangulated domain.** The blue region represents a single piece, where dark blue triangles indicate boundary elements requiring force term calculations. Light blue triangles represent internal elements that can be ignored in coarse layers since their net forces sum to zero.

5.1.2 Piece-wise Linear Approach. For our piece-wise linear approach, we discretize the object using a hierarchy of tetrahedra, where each coarse layer employs elements twice the size of its predecessor, yielding an approximate 8:1 reduction in vertex count between successive levels. This approach requires processing the complete set of finest elements across hierarchy levels.

We extend the coarse tetrahedral mesh to maximize coverage of fine tetrahedra, with special handling for boundary cases. For regions falling outside the coarse domain, we remove negative barycentric weights and renormalize. While this approximation could theoretically affect convergence, our experiments show these edge cases are rare and effectively addressed by finer resolution levels in our hierarchical solver structure.

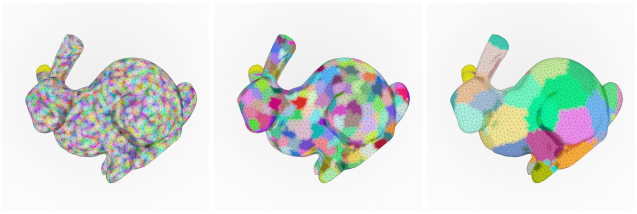


Fig. 4. Hierarchy visualization of our multigrid structure on the Stanford bunny model. Left: finest layer with individual points. Middle: layer 1 with initial patch merging. Right: layer 2 with larger consolidated patches. The gold region indicates fixed boundary constraints.

5.2 Prolongation and Restriction Operators

5.2.1 Constant Galerkin Formulation. For a cluster of tetrahedra with a residual function, the restriction operator in our piece-wise constant formulation is:

$$\boldsymbol{f}_c(\boldsymbol{x}) = \boldsymbol{P}^\top \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{P}^\top \boldsymbol{M}(\boldsymbol{x} - \hat{\boldsymbol{x}}) + \boldsymbol{P}^\top \sum_{i \in S} \nabla E_i(\boldsymbol{x}). \tag{17}$$

The piece-wise constant operator aggregates forces across elements, exploiting the principle that internal forces sum to zero. By ignoring internal elements and retaining only boundary elements, we achieve computational efficiency in coarse layers. However, this optimization may result in slower convergence compared to linear interpolation, potentially requiring additional iterations.

While piece-wise constant coefficients can be used with skinning space coordinates as in SGM, the introduced terms prevent us from ignoring internal elements, thus negating the computational advantages.

5.2.2 Linear Galerkin Formulation. The prolongation operator is constructed via barycentric interpolation between the finest layer and each coarse layer. To optimize the computational complexity inherent in managing four coarse parents per vertex and four vertices per tetrahedron, we implement a precomputation and merging strategy for element coarse parents.

5.3 Method Component Analysis

5.3.1 Hierarchy Configuration. Our hierarchy construction establishes direct connections between the finest layer and each coarse layer. While traditional 3D problems typically use a coarsening factor of 8, our experiments demonstrate that a scale factor of 32 provides an optimal balance between computational efficiency and solution accuracy. We therefore adopt 32 as our standard scaling factor across most experiments.

Figure 4 illustrates this hierarchy structure using a 50K vertex Stanford bunny model decomposed into three distinct layers. The left image shows the finest layer where each point is individually represented. The middle image displays the first coarsened layer (layer 1), where points are merged into small patches. The rightmost image presents the second coarsened layer (layer 2), featuring larger consolidated patches. The gold-colored region in each image indicates the fixed boundary constraints of the model.

5.3.2 Multigrid Cycle Strategies. Our formulation enables straightforward implementation of traditional cycle patterns (V-cycle, F-cycle, W-cycle) since each coarse layer computes directly from the finest layer. However, our experiments demonstrate that a simple coarse-to-fine progression yields the best efficiency. This finding aligns with optimization principles where smaller, more frequent steps typically outperform fewer, larger steps. We achieve better results by increasing the number of sub-timesteps, each with a single coarse-to-fine pass, rather than using complex cycle patterns.

5.3.3 Smoothing Pass Optimization. Following similar reasoning to our cycle strategy, we found that multiple pre-smooth or post-smooth passes within each level effectively function as additional iterations. Our experiments confirm that single smoothing passes with more frequent sub-timesteps achieve lower error with equivalent computational cost.

5.4 Complete Algorithm

Here, we present our complete multigrid algorithm. Algorithm 4 outlines our full multigrid method with full grid hierarchy and FAS formulation. Our method is notably concise, requiring only a simple coarse-to-fine loop to process the multilevel grid hierarchy. This simplicity stems from our direct FAS design that connects the finest grid directly to each coarse grid, eliminating the need for complex intergrid transfers found in traditional multigrid approaches.

ALGORITHM 4: Full Multigrid FAS with Vertex Block Jacobi

```
Initialize multigrid hierarchy ;
Initialize restriction and prologation weights P_l for every layer l;
for each frame do
    for s = 1 to sub-timesteps do
        Initial guess x = x_n + v_n Δt + g Δt^2, here x_n and v_n are
          position and velocity of last sub-timestep ;
        if Is collision pass then
            Collect collision pairs ;
        for layer = coarse to fine do
            for each valid element e do
                for related coarse point i to element e do
                    Compute the diagonal Hessian blocks H_{c,i} and
                      gradients g_{c,i} with our matrix-free method ;
                    Add the H_{c,i} and g_{c,i} to point i ;
            Solve coarse problem Δx_c ← −(diag(H_c)^{-1} g_c ;
            Apply correction x ← x + P_l Δx_c ;
        Update new positions x_{n+1} = x and velocities v_{n+1} = (x − x_n)/Δt;
```

6 Evaluation

This section presents a comprehensive evaluation of our method. We begin by describing our evaluation methodology and rationale to ensure clarity and reproducibility. Our experimental results are organized as follows: First, we directly compare our approach with state-of-the-art methods to demonstrate its efficacy. Second, we provide detailed single-threaded CPU comparisons to analyze the computational complexity of different approaches. Finally, we subject

our method to extreme test cases and diverse scene configurations to validate its stability and versatility across a wide range of simulation scenarios. Detailed simulation parameters and computational costs are presented in Table 1.

Termination Error. Traditional approaches to measuring convergence typically employ relative energy or residual errors. However, these metrics inadequately capture the full complexity of simulation accuracy. They overlook cumulative errors during simulation, particularly in implementations with fixed iteration counts, and fail to reflect the accumulation of simulation errors that affect visual accuracy. We therefore adopt an absolute error metric:

$$\epsilon(\boldsymbol{x}) = \frac{1}{\sqrt{N}}|(\boldsymbol{x} - \hat{\boldsymbol{x}}) + \boldsymbol{M}^{-1}\Delta t^2 \nabla E(\boldsymbol{x})|, \tag{18}$$

where N represents the number of points.

Performance Measurement Methodology. Performance measurement, particularly computational cost evaluation, is critical for fair comparisons. Since implementation details (especially in GPU parallel implementations) can significantly influence results, we followed specific principles to ensure fairness. For GPU comparisons, which are most relevant to real-world applications, we compared our optimized GPU implementation against previously released optimized versions of existing methods to demonstrate efficiency gains. Additionally, we conducted single-threaded CPU comparisons to eliminate parallelism variables and provide a clearer assessment of the fundamental computational requirements of each method.

Time Step Selection. Timestep selection proves crucial for overall solver performance. Our experiments corroborate the findings in [Macklin et al. 2019], showing that converting iterations to substeps yields superior accuracy. For standard Newton methods requiring explicit Hessian matrix construction and linear system solutions, larger timesteps remain computationally advantageous due to the high cost of matrix assembly. However, our matrix-free approach fundamentally alters this trade-off. Without the overhead of explicit matrix construction, smaller timesteps consistently outperform larger ones. Our experiments with fixed termination error criteria reveal an optimal timestep selection for the smoother, as illustrated in Figure 5. In our experiments, we use small time steps to achieve best performance.

6.1 Performance Evaluation with Previous Methods

We conducted comprehensive comparisons with publicly available implementations of state-of-the-art methods, specifically SGM [Xian et al. 2019] and VBD [Chen et al. 2024], to ensure a fair and rigorous validation of our approach. Due to compatibility constraints, we compared with SGM using an NVIDIA RTX 2070 GPU (as SGM's code requires CUDA 10, incompatible with newer GPUs), while VBD comparisons were performed on an RTX 4090 GPU using their CUDA 11.8 implementation. In the following sections, we present separate evaluations against both SGM and VBD to demonstrate our method's effectiveness. An important note regarding methodology: although all three methods (ours, SGM, and VBD) can benefit from Chebyshev acceleration, we deliberately disabled this optimization

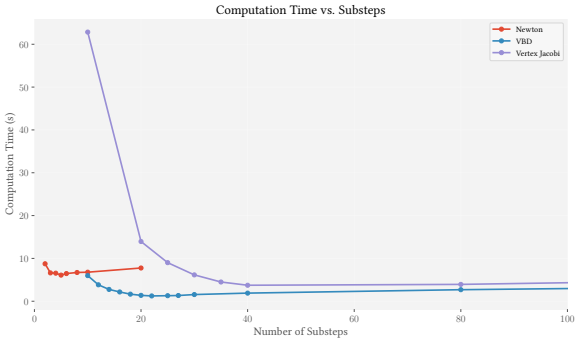


Fig. 5. **Impact of timestep selection on solver performance.** The horizontal axis represents the number of sub-timesteps per frame, while the vertical axis shows the average computation time (in seconds) needed to simulate one frame. The V-shaped curve reveals an optimal sub-timestep count that balances computational efficiency: increasing sub-timesteps initially improves performance dramatically, but beyond the optimal point, additional sub-timesteps lead to gradually increasing computational costs. This demonstrates the trade-off between convergence rate and sub-timesteps count in our matrix-free framework. We use a bunny drape scene with 20K particles to test the effect of time step selection.

in all implementations to eliminate any potential bias from parameter selection, ensuring a more equitable comparison of the core algorithms. Our test scenario is illustrated in Figure 6.

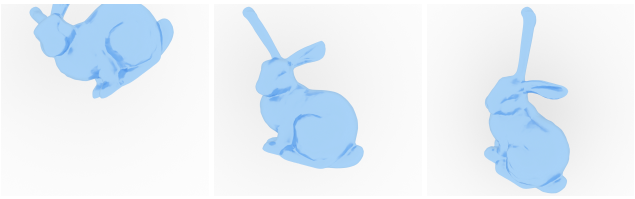


Fig. 6. The bunny drape test scene (200K vertices) used in our performance evaluations, shown at frames 0, 10, and 20 to illustrate the simulation dynamics.

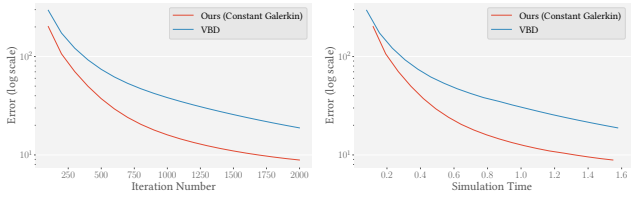
Comparison with VBD. To thoroughly evaluate our method against VBD, we conducted two distinct experiments: one focusing on convergence analysis within a single sub-timestep across multiple iterations, and another examining performance efficiency during practical simulation using fixed small steps with just one iteration per sub-timestep.

For our convergence analysis, we used the bunny drape scene to observe the behavior during the first sub-timestep. Figure 7a presents both the error-iterations relationship (left) and the error-time performance (right), clearly demonstrating that our method performs better than VBD. As shown in the plots, our method achieves lower error values with fewer iterations and reaches convergence more rapidly in terms of computational time.

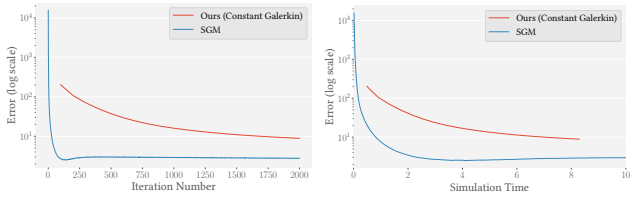
For our efficiency evaluation, we implemented both methods using 100 sub-timesteps with a single iteration per sub-timestep, and monitored their performance throughout the entire simulation. Figure 7c illustrates the computational cost (right) and error metrics

Table 1. **Simulation Parameters and Performance.** Parameters and computation time for various test scenes. #Points and #Tets represent mesh resolution in number of vertices and tetrahedral elements. E is Young's modulus (Pa) indicating material stiffness, with ν being Poisson's ratio and ρ the material density (kg/m³). Damping coefficient controls energy dissipation rate. Contact parameters k_c (stiffness, Pa) and μ_c (friction coefficient) govern collision response. $s_{\text{layer}}|N_{\text{layer}}$ indicates scale between layers and number of layers. Substeps show time step subdivisions per frame, while Iterations represent solver iterations per sub-timestep. Time Per Frame shows computation time for each frame with 1/60s time clip. Dashes ("-") indicate parameters varying across experiments as detailed in the corresponding sections or not used. Frames shows the total frames number of simulation, and the precision shows the float number precision we use during simulation.

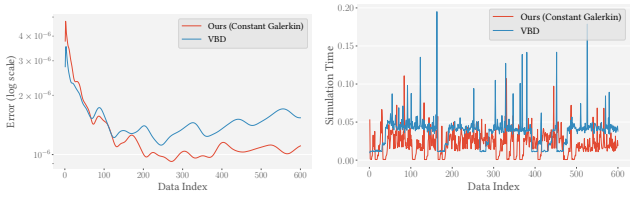
Scene	#Points	#Tets	E	ν	ρ	Damping	k_c	μ_c	$s_{\text{layer}}\|N_{\text{layer}}$	Sub-timesteps	Iterations	Precision	Time Per Frame	Frames
Bunny Drape (200K Points)	205K	1.27M	10^6	0.4	1	0	-	-	8 | full	-	-	float	-	-
Bunny Stretch	44K	266K	10^6	0.4	2600	2^{-5}	-	-	32 | full	100	1	double	0.13 s	120
Cube	26K	162K	10^6	0.4	2600	2×10^{-5}	-	-	32 | full	100	1	double	0.068 s	60
Teapot	22K	88K	10^6	0.4	2600	8×10^{-6}	-	-	32 | full	20	1	double	0.008 s	120
Armardillo Flat	15K	55K	10^6	0.4	2600	8×10^{-6}	-	-	32 | full	100	1	double	0.032 s	120
Beam Stretch	537	1.7K	10^6	-	2600	2×10^{-5}	-	-	32 | full	100	1	double	0.002 s	120
Buddha	163K	920K	10^8	0.4	2600	10^{-8}	-	-	32 | full	800	1	double	3.1 s	120
Many Armardillos	142K	574K	10^6	0.4	2600	2×10^{-7}	10^6	0.3	32 | full	200	1	double	2.5 s	600
Squishy Ball	352K	1.04M	10^6	0.4	2600	2×10^{-7}	10^6	0.3	32 | 2	400	1	double	1.41 s	600
Dragon Drop (1M Points)	1.15M	4.88M	10^6	0.4	2600	10^{-7}	10^6	0.3	32 | full	400	1	float	1.15 s	100
Bunny Drape (1M Points)	1.19M	7.52M	10^6	0.4	2600	10^{-8}	-	-	32 | full	400	1	float	1.6 s	600



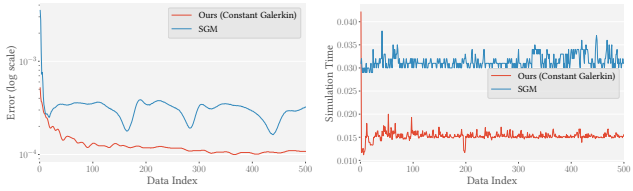
(a) VBD: Single-substep convergence analysis



(b) SGM: Single-substep convergence analysis



(c) VBD: Multi-frame performance comparison



(d) SGM: Multi-frame performance comparison

Fig. 7. Comprehensive performance comparison with VBD and SGM methods. (a) VBD single-substep convergence showing our method achieves lower error with fewer iterations and faster computational time. (b) SGM single-substep convergence analysis demonstrating baseline performance metrics. (c) VBD multi-frame performance comparison showing our method maintains consistently lower error values and reduced computational overhead throughout the simulation. (d) SGM multi-frame performance using 300 sub-timesteps for our method, demonstrating approximately 50% lower error while requiring only half the computational resources compared to SGM.

(left) for all sub-timesteps across the simulation timeline, showing that the error and time cost are both better than VBD. The consistently lower error values and reduced computational overhead demonstrate our method's superior efficiency in practical simulation scenarios.

Comparison with SGM. We evaluated SGM's implementation using identical scene configurations and simulation parameters. Similar to our VBD comparison, we conducted two types of experiments: single-substep convergence analysis and practical simulation with small steps.

In the single-substep convergence analysis shown in Figure 7b, SGM demonstrates superior performance in terms of both convergence rate and computational efficiency. However, several important factors influence these results. First, our method was specifically optimized for RTX 4090 GPUs but run without adaptation on the RTX 2070 used in this comparison. In contrast, the original SGM implementation leverages CUDA 10.0's cuBLAS libraries, which are specifically optimized for the Turing architecture of the RTX 2070. Despite these architectural advantages for SGM, our method still achieves better performance in practical multi-frame simulations, as shown in the second experiment, even without platform-specific optimizations.

For our practical efficiency evaluation, we configured SGM with 100 sub-timesteps per frame and one multigrid iteration per sub-timestep as a baseline. Our analysis revealed that one iteration of

our constant Galerkin multigrid method is substantially less computationally intensive than SGM's iteration, which enables our method to benefit more from smaller time steps. We examined a 300 sub-timestep configuration for our method (Figure 7d), which reduced the error by approximately 50% compared to SGM while requiring only half the computational resources. This clearly demonstrates that our method outperforms SGM in realistic application scenarios, offering a superior balance between accuracy and computational efficiency.

6.2 Performance Evaluation on Single-Threaded CPU

In the last section, we compared GPU performance between different methods. Here, we evaluate single-threaded CPU performance to demonstrate that our approach not only excels in parallel environments but also reduces overall computational cost and achieves superior performance in sequential implementations. For fair comparison, we use straightforward implementations for all methods. We compare the performance of six methods:

- **Jacobi-Preconditioned Conjugate Gradient (PCG):** A standard iterative solver widely adopted in previous works like SGM, MAS, and AMG. We configure it with adaptive Newton iterations and a fixed relative error tolerance of 5×10^{-3}, matching the settings used in AMG [Tamstorf et al. 2015] for fair comparison.
- **AMGCL:** An algebraic multigrid library representing traditional multigrid implementations, providing a benchmark against established frameworks.
- **Matrix-Free VBD:** VBD [Chen et al. 2024] method enhanced with our matrix-free optimization to accelerate matrix computations.
- **Matrix-Free Jacobi (Our Smoother):** Our baseline smoother implementation described in Section 4, demonstrating the fundamental performance benefits of the matrix-free approach.
- **Piecewise Constant Multigrid (Our Method):** Our implementation of the piecewise constant Galerkin formulation detailed in Section 5.
- **Piecewise Linear Multigrid (Our Method):** Our implementation featuring piecewise linear Galerkin formulation from Section 5.

6.2.1 Performance on Different Resolution. A critical measure of multigrid method effectiveness is its scalability across different resolutions. Our comparative analysis examines performance metrics with detailed results shown in Figure 8.

As demonstrated in the left plot, our multigrid methods significantly outperform traditional approaches in CPU computation time across all resolutions. The constant Galerkin variant achieves the lowest computation time among all methods, with the performance gap widening as resolution increases. At 1M elements, our method demonstrates more than 6x speedup over Jacobi-PCG. The middle plot reveals an important convergence behavior: while the Jacobi smoother requires increasingly more sub-timesteps at higher resolutions, our multigrid methods maintain nearly constant sub-timestep counts regardless of resolution. This stability is crucial for handling complex, high-resolution meshes efficiently. The iteration count comparison (right plot) further illustrates our method's efficiency. Both Jacobi-PCG and AMGCL require significantly more inner iterations, with their counts growing rapidly at higher resolutions. In contrast, our multigrid approach maintains relatively stable iteration requirements across all test cases.

Table 2 quantifies these improvements, showing our constant Galerkin multigrid achieves speedups ranging from 2.15x to 6.90x over the baseline Jacobi-PCG. Notably, our method's advantage becomes more pronounced at higher resolutions, reaching a 6.90x speedup for meshes with 1M elements.

Table 2. **Performance comparison across different mesh resolutions, with Jacobi-PCG as the baseline.** Our multigrid method demonstrates consistent speedup across all resolutions, achieving particularly significant improvements for high-resolution simulations. At 1M vertices, our method achieves a 6.90x speedup over the baseline, highlighting its efficiency for large-scale simulations.

Resolution	2K	10K	20K	100K	200K	500K	1M
Jacobi-PCG	1.0	1.0	1.0	1.0	1.0	1.0	1.0
AMGCL	0.47	0.65	0.50	0.69	0.69	0.60	0.88
VBD	1.43	1.66	1.70	1.82	1.92	1.95	3.89
Jacobi-Smoother	1.73	1.74	2.02	2.56	2.57	1.52	3.74
Ours (Constant)	**2.15**	**2.49**	**2.71**	**4.25**	4.28	**3.36**	**6.90**
Ours (Linear)	1.71	2.15	2.46	3.72	**4.68**	2.82	5.46

6.2.2 Performance Across Stiffness Ranges. Figure 9 presents our performance analysis across varying stiffness values from 10^3 to 10^8. As shown in the left plot, our methods consistently achieve the lowest computation times across all stiffness ranges. The middle plot reveals that while all methods require more sub-timesteps as stiffness increases, our multigrid approaches maintain the lowest sub-timestep counts across the spectrum. The right plot demonstrates that our methods require significantly fewer inner iterations than both Jacobi-PCG and AMGCL across all stiffness values.

Table 3 quantifies these improvements, showing our constant Galerkin formulation achieves up to 6.19x speedup over the baseline at low stiffness, while maintaining robust 2.45x-2.86x improvements even at the highest stiffness values. This consistent performance advantage across diverse material properties demonstrates the versatility and efficiency of our approach.

Table 3. **Performance comparison across varying material stiffness values, with Jacobi-PCG as the baseline.** Experiments conducted on the bunny drape scene with 200K vertices. Our multigrid methods demonstrate superior performance across all stiffness ranges. At lower stiffness, our constant formulation achieves up to 6.19x speedup, while at extreme stiffness (10^8), both formulations maintain robust performance advantages.

Stiffness	10^3	10^4	10^5	10^6	10^7	10^8
Jacobi-PCG	1.0	1.0	1.0	1.0	1.0	1.0
AMGCL	0.86	0.73	0.51	0.69	0.72	0.39
VBD	3.00	2.63	2.30	1.82	1.83	1.70
Jacobi-Smoother	4.75	3.91	3.02	2.56	1.85	2.17
Ours (Constant)	**6.19**	**5.30**	**4.09**	**4.25**	2.57	2.45
Ours (Linear)	3.18	2.29	3.57	2.79	**2.68**	**2.86**

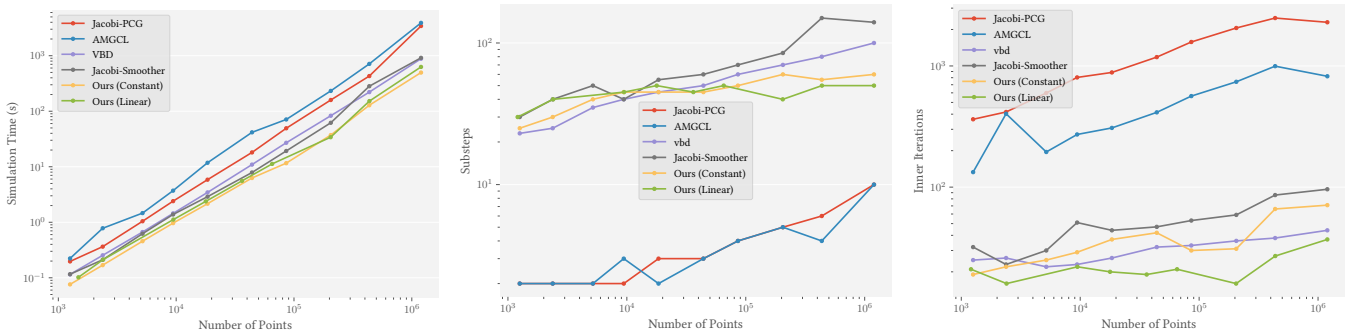


Fig. 8. **Performance analysis across different mesh resolutions with material stiffness of** 1×10^6**.** Left: CPU computation time comparison, demonstrating substantial speedups achieved by our multigrid method over traditional approaches. Middle: Required sub-timestep counts across resolutions, highlighting our multigrid method's ability to maintain consistent sub-timestep numbers while the Jacobi smoother exhibits increasing requirements at higher resolutions. Right: Per-frame inner iteration counts, showing our matrix-free multigrid approach requires significantly fewer iterations than both Jacobi-PCG and AMGCL, with superior scaling at high resolutions.

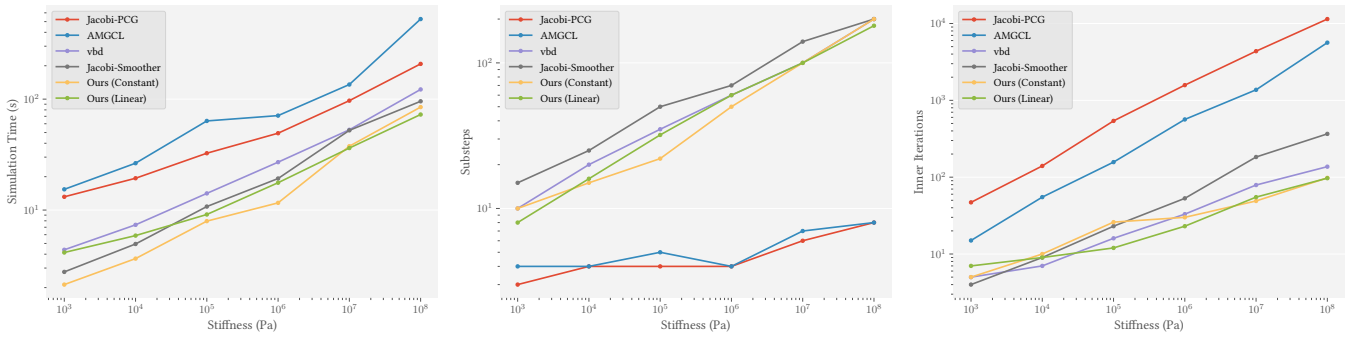


Fig. 9. **Performance comparison across material stiffness values ranging from** 10^3 **to** 10^8**.** Left: CPU computation time comparison, demonstrating that our multigrid methods (constant-weight and linear-weight) significantly outperform traditional approaches, with the constant-weight variant showing particular efficiency at higher stiffness values. Middle: Sub-timestep requirements across stiffness values, illustrating that while the Jacobi smoother requires increasingly more iterations at high stiffness, our multigrid method maintains consistent performance. Right: Inner iteration counts per frame, where our methods demonstrate superior efficiency compared to Jacobi-PCG and AMGCL. Although our constant-weight variant requires more iterations than its linear-weight counterpart, it achieves better overall performance through reduced per-iteration computational cost. At extreme stiffness (10^8), the robustness of our approach becomes particularly evident as AMGCL's performance degrades under default parameters.

6.3 Demonstration Cases

Having validated our method through comparative analysis and extreme scenario testing, we present additional examples demonstrating its stability across varying material properties, complex collision scenarios, and large-scale simulations with up to one million vertices.

Materials with Varying Poisson Ratios. Our method naturally accommodates different Poisson ratios by simply adjusting the corresponding parameter in the Stable Neo-Hookean model. Figure 10 demonstrates this through a beam stretching experiment with Poisson ratios of 0.1, 0.2, 0.4, and 0.49, handled stably without parameter tuning.

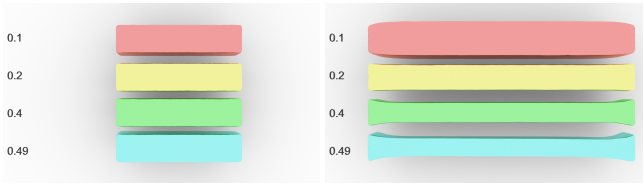


Fig. 10. **Beam stretch** experiment with varying Poisson ratios (0.1, 0.2, 0.4, 0.49). Left: initial configuration; Right: stretched state showing different lateral contractions corresponding to the Poisson ratios.

High-Stiffness Materials. We simulate a tilted Buddha model (163K vertices) dropping onto a rigid surface using highly stiff material parameters (E = 10^8 Pa, $\nu = 0.4$) and an initial 20-degree tilt. Our solver maintains stability using 800 sub-timesteps with only 1 multigrid iteration per step (Figure 11), highlighting our method's efficiency with high-stiffness materials and impact dynamics.

Fig. 11. **Buddha.** Simulation sequence of a Buddha model (E = 10^8 Pa) dropping and impacting a rigid surface, stable with 800 sub-timesteps and 1 multigrid iteration per step.

Inter-object Collision. We simulate eight Armadillo models (142K total vertices) dropping into a confined box, each using our multigrid method within a unified collision detection framework. The simulation uses 200 sub-timesteps with E = 10^6 Pa. Figure 12 shows our method effectively handling multiple deformable objects with extensive collision interactions.

Self-collision Handling. We test self-collision handling with a highly deformable squishy ball impacting a rigid surface. Figure 13 shows the ball's considerable deformation creating numerous self-contact regions that our method resolves efficiently and stably. This example also demonstrates our method's capability to handle non-uniform tessellations, as the tetrahedral elements in the fur region have significantly different sizes compared to those in the ball interior, showcasing the robustness of our multigrid approach across varying element scales.

Dragon Drop with 1M points. We simulate a million-vertex elastic dragon (E = 10^7 Pa) impacting a rigid surface from a 1m/s initial velocity. Our method requires only 200 sub-timesteps with just 1 iteration per sub-timestep to achieve stable and visually plausible results, as shown in Figure 14.

Bunny Drape with 1M points. We simulate a bunny model with its right ear fixed (E = 10^6 Pa). For the million-vertex case, our constant Galerkin multigrid method achieves visually accurate results with 400 sub-timesteps and only 1 iteration per sub-timestep, demonstrating the scalability of our approach.

7 Conclusion

We present an efficient multigrid solver that combines the Full Approximation Scheme with Galerkin formulation for unstructured tetrahedral meshes. Our approach introduces a matrix-free vertex block Jacobi smoother that overcomes the previously prohibitive computational costs of dense coarse matrices in Galerkin methods. This framework achieves up to 6.9× speedup over standard Jacobi-PCG approaches and outperforms recent state-of-the-art methods including SGM and VBD across various metrics. Our implementation supports both piecewise constant and piecewise linear Galerkin formulations, with the constant variant providing optimal performance for most scenarios. Comprehensive experiments demonstrate our method's robustness across diverse challenges: extreme deformations exceeding 2000% stretching, severely distorted initial configurations, materials with varying Poisson ratios (0.1-0.49), high-stiffness materials (up to 10^8 Pa), and complex collision interactions. Our GPU-optimized implementation achieves performance of approximately one second per frame for simulations involving one million tetrahedral vertices, making high-fidelity physics simulation practical for graphics applications.

7.1 Limitations and Future Work

A key limitation in our current implementation concerns the selection of Jacobi relaxation parameters. While we explored both fixed-step and line search strategies, our GPU implementation revealed that line search operations introduce reduction and synchronization overhead that impacts performance. To maximize computational efficiency, we opted for fixed step lengths–0.25 for coarse layers and 0.5 for fine layers. Although this approach ensures stability across our experiments, it may not achieve optimal convergence rates. Our line search experiments indicate that coarse layer step lengths often benefit from significantly larger values, sometimes reaching 2.0 or 4.0. This suggests promising directions for future research in developing adaptive step length selection methods that maintain GPU efficiency while accelerating convergence.

Additionally, while collision handling was not the primary focus of this work, our current implementation employs a simplified collision resolution strategy. Future work will explore the integration of more sophisticated collision handling techniques to enhance the robustness and accuracy of dynamic simulations. Another limitation stems from the inherent characteristics of multigrid methods when applied to non-manifold geometries. Since multigrid approaches generally exhibit reduced effectiveness on non-manifold structures, our method similarly faces challenges in handling such complex topologies.

Several other areas present opportunities for improvement. The convergence rate of our Jacobi smoother could be enhanced through alternative smoothers that better utilize local information, potentially outperforming our current Jacobi-preconditioned gradient descent approach. Given the heuristic nature of our graph clustering algorithm, more sophisticated clustering approaches could yield better domain decomposition and improve overall performance.

References

Mridul Aanjaneya, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. 2017. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–12.

David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of ACM SIGGRAPH 1998.* 43–54.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics. *ACM Transactions on Graphics* 33, 4 (2014), 1–11.

Achi Brandt. 1977. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation* 31, 138 (1977), 333–390.

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

Nuttapong Chentanez and Matthias Müller. 2011. Real-time Eulerian water simulation using a restricted tall cell grid. In *ACM Siggraph 2011 Papers.* 1–10.

Christian Dick, Marcus Rogowsky, and Rüdiger Westermann. 2015. Solving the fluid pressure Poisson equation using multigrid—evaluation and improvements. *IEEE transactions on visualization and computer graphics* 22, 11 (2015), 2480–2492.

Florian Ferstl, Rüdiger Westermann, and Christian Dick. 2014. Large-scale liquid simulation on adaptive hexahedral grids. *IEEE transactions on visualization and computer graphics* 20, 10 (2014), 1405–1417.

Theodore F Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M Teran. 2015. Optimization integrator for large time steps. *IEEE transactions on visualization and computer graphics* 21, 10 (2015), 1103–1115.

Joachim Georgii and Rüdiger Westermann. 2006. A multigrid framework for real-time simulation of deformable bodies. *Computers & Graphics* 30, 3 (2006), 408–415.

Fig. 12. **Armadillo Drop.** Eight Armadillo models (142K vertices) interacting in a confined box, showing initial configuration, mid-fall interactions, and final settling state with E = 10^6 Pa.

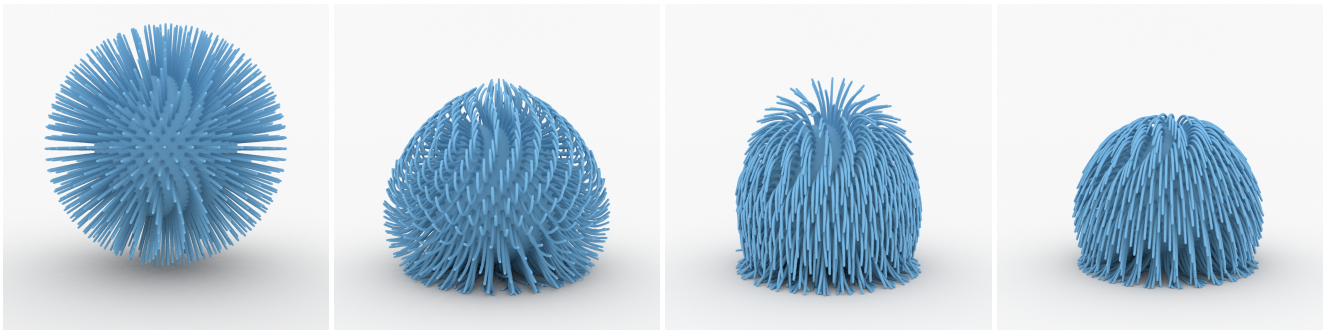


Fig. 13. **Squishy Ball.** A deformable ball impacting a rigid surface, showing initial descent, impact deformation, maximum compression, and final rest state.

Fig. 14. **Dragon Drop.** Million-vertex dragon impacting a rigid surface, showing initial state, impact deformation, and rebound with 200 sub-timesteps and 1 iteration per sub-timesteps.

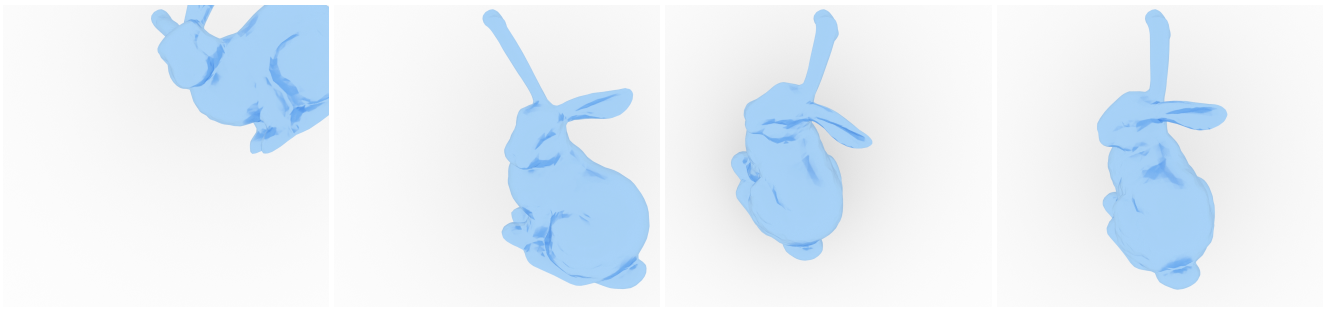


Fig. 15. **Bunny Drape with 1M points.** Fixed-ear bunny deforming under gravity, showing initial state, early deformation, mid-simulation, and equilibrium, using 400 sub-timesteps and 1 iteration per sub-timestep.

Markus Hegland and Paul E Saylor. 1992. Block jacobi preconditioning of the conjugate gradient method on a vector processor. *International journal of computer mathematics* 44, 1-4 (1992), 71–89.

Inyong Jeon, Kwang-Jin Choi, Tae-Yong Kim, Bong-Ouk Choi, and Hyeong-Seok Ko. 2013. Constrainable multigrid for cloth. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 31–39.
Theodore Kim and David Eberle. 2020. Dynamic deformables: implementation and production practicalities. In *ACM SIGGRAPH 2020 Courses*. 1–182.
Yongjoon Lee, Sung-eui Yoon, Seungwoo Oh, Duksu Kim, and Sunghee Choi. 2010. Multi-resolution cloth simulation. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 2225–2232.
Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy R Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. 2020. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.* 39, 4 (2020), 49.
Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. 2021. Surface multigrid via intrinsic prolongation. *ACM Transactions on Graphics* 40, 4 (2021), 13 pages.
Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. 2013. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–7.
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*. 1–7.
Ahmed H Mahmoud, Serban D Porumbescu, and John D Owens. 2021. RXMesh: a GPU mesh data structure. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–16.
Aleka McAdams, Eftychios Sifakis, and Joseph Teran. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids.. In *Symposium on Computer Animation*, Vol. 65. 74.
Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient elasticity for character skinning with contact and collisions. In *ACM SIGGRAPH 2011 papers*. 1–12.
Alexandre Mercier-Aubin and Paul G Kry. 2024. A Multi-layer Solver for XPBD. In *Computer Graphics Forum*. Wiley Online Library, e15186.
Matthias Müller. 2015. Hierarchical position-based dynamics. (03 2015).
Maxim Naumov, Marat Arsaev, Patrice Castonguay, Jonathan Cohen, Julien Demouth, Joe Eaton, Simon Layton, Nikolay Markovskiy, István Reguly, Nikolai Sakharnykh, et al. 2015. AmgX: A library for GPU accelerated algebraic multigrid and preconditioned iterative methods. *SIAM Journal on Scientific Computing* 37, 5 (2015), S602–S626.
Miguel A Otaduy, Daniel Germann, Stephane Redon, and Markus Gross. 2007. Adaptive deformations with fast tight bounds. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 181–190.
Liangwang Ruan, Bin Wang, Tiantian Liu, and Baoquan Chen. 2024. Minnie: a mixed multigrid method for real-time simulation of nonlinear near-incompressible elastics. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–15.
Han Shao, Libo Huang, and Dominik L Michels. 2022. A fast unsmoothed aggregation algebraic multigrid framework for the large-scale simulation of incompressible flow. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–18.
Frédéric Sicot, Guillaume Puigt, and Marc Montagnac. 2008. Block-Jacobi implicit algorithms for the time spectral method. *AIAA journal* 46, 12 (2008), 3080–3089.
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 Transactions on Graphics (TOG)* 34, 6 (2015), 1–13.
Ty Trusty, Danny Kaufman, and David IW Levin. 2022. Mixed variational finite elements for implicit simulation of deformables. In *SIGGRAPH Asia 2022 Conference Papers*. 1–8.
Huamin Wang. 2015. A Chebyshev Semi-Iterative Approach for Accelerating Projective and Position-based Dynamics. *ACM Transactions on Graphics* 34 (10 2015), 1–9. doi:10.1145/2816795.2818063
Huamin Wang and Yin Yang. 2016. Descent methods for elastic body simulation on the GPU. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–10.
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.
Xiaokun Wang, Yanrui Xu, Sinuo Liu, Bo Ren, Jirí Kosinka, Alexandru C Telea, Jiamin Wang, Chongming Song, Jian Chang, Chenfeng Li, et al. 2024. Physics-based fluid simulation in computer graphics: Survey, research trends, and challenges. *Computational Visual Media* 10, 5 (2024), 803–858.
Zhendong Wang, Longhua Wu, Marco Fratarcangeli, Min Tang, and Huamin Wang. 2018. Parallel multigrid for nonlinear cloth simulation. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 131–141.
Daniel Weber, Johannes Mueller-Roemer, André Stork, and Dieter Fellner. 2015. A Cut-Cell Geometric Multigrid Poisson Solver for Fluid Simulation. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 481–491.
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.
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.
Jiayi Eris Zhang, Jérémie Dumas, Yun Fei, Alec Jacobson, Doug L James, and Danny M Kaufman. 2022. Progressive simulation for cloth quasistatics. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–16.
Jiayi Eris Zhang, Jérémie Dumas, Yun Fei, Alec Jacobson, Doug L James, and Danny M Kaufman. 2023. Progressive Shell Qasistatics for Unstructured Meshes. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–17.
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.
Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Transactions on Graphics (TOG)* 29, 2 (2010), 1–18.

A Other Terms for Deformable Simulation

Self Collision. We handle collisions by incorporating a collision force term into our solving Equation (1). The collision force is derived from the potential

$$\Psi_{\text{col}}(\boldsymbol{x}) = \frac{1}{2}\kappa d^2, \quad d = \max(0, (\boldsymbol{x}_a - \boldsymbol{x}_b)\cdot\hat{\boldsymbol{n}}), \tag{19}$$

where κ denotes the collision stiffness, $\hat{\boldsymbol{n}}$ is the normal vector of collison surface, and $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ represent the collision points. By employing a common simplification that assumes a constant direction $\hat{\boldsymbol{n}}$, we derive the gradient and Hessian for $\boldsymbol{x}_a$ as

$$\nabla_a\Psi_{\text{col}}(\boldsymbol{x}) = \kappa d\hat{\boldsymbol{n}}, \quad \nabla_a^2\Psi_{\text{col}}(\boldsymbol{x}) = \kappa\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^\top. \tag{20}$$

For the stencil vertices $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$, both $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ can be expressed as linear combinations of these vertices. Maintaining our assumption of constant linear weights, we express the Hessian matrix as:

$$\nabla\Psi_{\text{col}}(\boldsymbol{x}) = \boldsymbol{\sigma}\otimes(\kappa d\hat{\boldsymbol{n}}), \quad \nabla^2\Psi_{\text{col}} = (\boldsymbol{\sigma}\boldsymbol{\sigma}^\top)\otimes(\kappa\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^\top). \tag{21}$$

Here, $\boldsymbol{\sigma}$ represents the weight vector derived from the linear combination weights of $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$.

- For vertex-face pairs, $\boldsymbol{x}_a$ corresponds to $\boldsymbol{x}_0$, and with barycentric weights $(1-u-v, u, v)$ for $\boldsymbol{x}_b$, we have $\boldsymbol{\sigma} = (1, u+v-1, -u, -v)$,
- For edge-edge pairs, with barycentric weights $(1-s, s)$ for $\boldsymbol{x}_a$ and $(1-t, t)$ for $\boldsymbol{x}_b$, we have $\boldsymbol{\sigma} = (1-s, s, 1-t, t)$.

Given the restriction operator $\boldsymbol{p}_i$ from Equation (10), the stencil gradient and Hessian contributions for coarse point i are:

$$\begin{aligned}(\boldsymbol{b}_c)_i &= (\boldsymbol{\sigma}\cdot\boldsymbol{w})\kappa d\hat{\boldsymbol{n}},\\ (\boldsymbol{H}_c)_i &= (\boldsymbol{\sigma}\cdot\boldsymbol{w})^2\kappa\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^\top.\end{aligned} \tag{22}$$

While these terms are computationally efficient, they differ from neo-Hookean energy in that we cannot precompute weights $\boldsymbol{w}$ for all coarse parents due to unknown collision pairs during initialization. However, since collision essentially acts as a boundary condition, we can precompute and sort the parent indices for each surface edge and face, then merge corresponding weights through index-sorted combination.

Friction. We implement the friction scheme described in [Chen et al. 2024; Li et al. 2020]. For identified collision pairs, we compute the relative collision velocity as $\boldsymbol{v}_c = (\boldsymbol{x}_a - \boldsymbol{x}_a^t) - (\boldsymbol{x}_b - \boldsymbol{x}_b^t)$, where $\boldsymbol{x}_a^t$ and $\boldsymbol{x}_b^t$ denote positions from the previous time step. Using a

transformation matrix $\boldsymbol{T}_c$, we project $\boldsymbol{v}_c$ onto the tangential collision plane to obtain $\boldsymbol{u}_c = \boldsymbol{T}_c^\top \boldsymbol{v}_c$. Utilizing the $\boldsymbol{\sigma}$ vector from our collision formulation, we express the friction terms as:

$$\begin{aligned} \nabla \Psi_{\text{friction}}(\boldsymbol{x}) &= \boldsymbol{\sigma} \otimes (-\mu_c \lambda_{c,i} \boldsymbol{T}_c \boldsymbol{f}_1(\|\boldsymbol{u}_c\|) \frac{\boldsymbol{u}_c}{\|\boldsymbol{u}_c\|}), \\ \nabla^2 \Psi_{\text{friction}}(\boldsymbol{x}) &= (\boldsymbol{\sigma}\boldsymbol{\sigma}^\top) \otimes (\mu_c \lambda_{c,i} \boldsymbol{T}_c \boldsymbol{f}_1(\|\boldsymbol{u}_c\|) \boldsymbol{T}_c^\top). \end{aligned} \tag{23}$$

Here, μ_c represents the friction coefficient, and $\lambda_{c,i} = \frac{\partial E_c}{\partial \boldsymbol{x}_i}$ denotes the signed magnitude for each position $\boldsymbol{x}_i$. The smooth transition function f_1 is defined as:

$$f_1(u) = \begin{cases} 2(\frac{u}{\Delta t \epsilon_v}) - (\frac{u}{\Delta t \epsilon_v})^2, & u < \Delta t \epsilon_v, \\ 1, & u \geq \Delta t \epsilon_v, \end{cases} \tag{24}$$

where ϵ_v controls the threshold between static and dynamic friction. The corresponding gradient and Hessian contributions are:

$$\begin{aligned} (\boldsymbol{b}_c)_i &= -(\boldsymbol{\sigma} \cdot \boldsymbol{w}) \mu_c \lambda c, i \boldsymbol{T}_c f_1(\|\boldsymbol{u}_c\|) \frac{\boldsymbol{u}_c}{\|\boldsymbol{u}_c\|}, \\ (\boldsymbol{H}_c)_i &= (\boldsymbol{\sigma} \cdot \boldsymbol{w})^2 \mu_c \lambda c, i \boldsymbol{T}_c f_1(\|\boldsymbol{u}_c\|) \boldsymbol{T}_c^\top. \end{aligned} \tag{25}$$

Manual Damping. We incorporate Rayleigh damping as described in [Chen et al. 2024] for controlled energy dissipation. This formulation seamlessly integrates with our gradient and Hessian computations.

B Larger-Scale Experiments

We tested our method on models with up to 1M vertices in Section 6. To further investigate whether our approach provides greater speedup ratios on larger-scale problems, we conducted additional experiments with 5M vertices. At this scale, GPU memory became insufficient, and several CPU methods also failed due to memory limitations. Through storage optimizations that trade time for space, we successfully ran 5M vertex bunny drape simulations on CPU, comparing Jacobi PCG and our Constant Galerkin method.

Due to the computational demands at this scale, we adjusted the simulation duration from 1/60s to 1/2400s to keep experiments tractable. Results showed that Jacobi PCG required 11623 s while our method needed only 1403 s, achieving a 8.3x speedup. This demonstrates that our multigrid approach maintains strong acceleration at larger scales. However, the additional memory management overhead introduces computational costs that do not purely reflect algorithmic performance, so we include these results in the appendix as reference data.

C Extreme Test Cases

Extreme Stretching. We tested the bunny model under three challenging stretching scenarios, where the model was elongated to more than 20 times its original length in Figure 16. The first scenario involved gradually stretching the bunny, the second initialized the model with stretch points and allowed it to deform back to rest state, and the third initialized with fixed stretch points, forcing the bunny to deform to the corresponding stretched configuration. These experiments demonstrate the stability of our method under extreme deformation conditions.

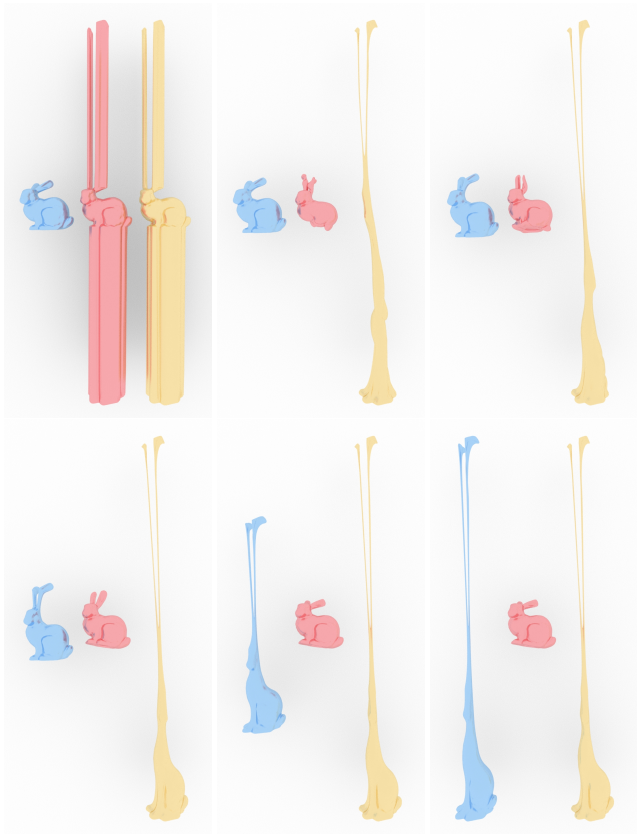


Fig. 16. Extreme stretching test on the bunny model showing frames from the simulation with progressive deformation, demonstrating our method's ability to handle extreme stretching conditions while maintaining stability.

Severely Distorted Initial Configurations. We conducted three types of tests with severely distorted initial configurations as shown in Figure 17. The first test replicated the expanded cube experiment from [Smith et al. 2018], but with initial vertex positions randomized within a volume twice the size of the rest shape. The second test, inspired by [Chen et al. 2024], randomly initialized the Utah teapot model with vertices positioned on a sphere surface. The third test flattened the armadillo model onto a surface as its initial configuration. All cases successfully recovered their intended shapes, validating our method's robust convergence properties even under highly challenging initial conditions.

D GPU Implementation

Our multigrid approach requires traversing full-resolution elements and summing their contributions to layer points for each layer. While element traversal naturally parallelizes, optimizing the reduce operator is crucial for GPU performance, particularly for higher layers where the small number of coarse points presents unique parallel efficiency challenges.

The reduction process accumulates both gradient vectors (3 components) and symmetric block Hessian matrices (6 unique components) per vertex. At the finest level, atomic operations are efficient

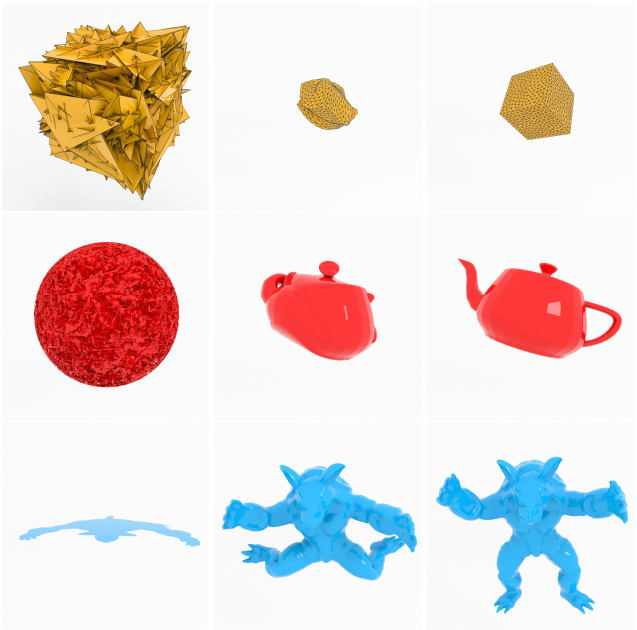


Fig. 17. Recovery from severely distorted initial configurations. Top row: cube with randomized vertices at twice the rest volume size. Middle row: teapot with vertices randomly placed on a sphere. Bottom row: armadillo model flattened to a surface. Columns show the initial configuration (left), intermediate state (middle), and final recovered shape (right).

due to balanced element-to-vertex ratios and minimal write conflicts. However, coarser levels experience significant thread contention due to increasing element-to-vertex ratios, rendering atomic operations inefficient.

We propose several optimization strategies, focusing on general approaches applicable across different hardware architectures rather than device-specific optimizations.

Morton Sorting. Our implementation assigns one tetrahedral element per thread. At the GPU block level, different elements can share points, allowing for efficient gathering operations using warp operators or shared memory, which significantly outperform global memory operations. To maximize block-level gathering, elements must be divided into blocks that share as many points as possible. While graph clustering algorithms like Lloyd k-means (discussed in Section 5) could achieve this, they struggle to maintain fixed patch sizes necessary for optimal GPU utilization. Instead, we employ Morton sorting, similar to [Wu et al. 2022]'s approach for multilevel preconditioner construction. This method sorts elements by spatial position and divides them into GPU blocks based on sorted indices. Common indices within blocks are processed using shared memory, with results written to global memory only after block operations complete. This strategy reduces global operations to 10-20% compared to individual element processing.

Grid Reduction. For coarse layers, atomic operations on global memory remain costly due to increased conflicts from fewer coarse points relative to block numbers. We address this by introducing intermediate global memory to store per-block point terms, followed by a separate kernel for grid reduction. The reduction patterns remain static throughout simulation, allowing pre-computation of parallel execution structures. Layer-specific optimizations are implemented: finer layers benefit from warp-level reduction due to fewer terms, while coarser layers require block-level reduction. We utilize the CUB library for these operations.

Spatial Hashing. GPU collision handling employs efficient spatial hashing through a two-phase approach: a broad phase detecting AABB bounding box intersected surface triangle pairs, and a narrow phase identifying vertex-face or edge-edge stencil intersections. Given our method's small timesteps, updating spatial hash structures every timestep would dominate elastic computation costs. We optimize by updating spatial hash every 10-20 timesteps, balancing computational efficiency with collision precision.