Second-order Stencil Descent for Interior-point Hyperelasticity

LEI LAN, The University of Utah, USA
MINCHEN LI, UCLA, USA
CHENFANFU JIANG, UCLA, USA
HUAMIN WANG, Style3D Research, China
YIN YANG, The University of Utah & Style3D Research, USA

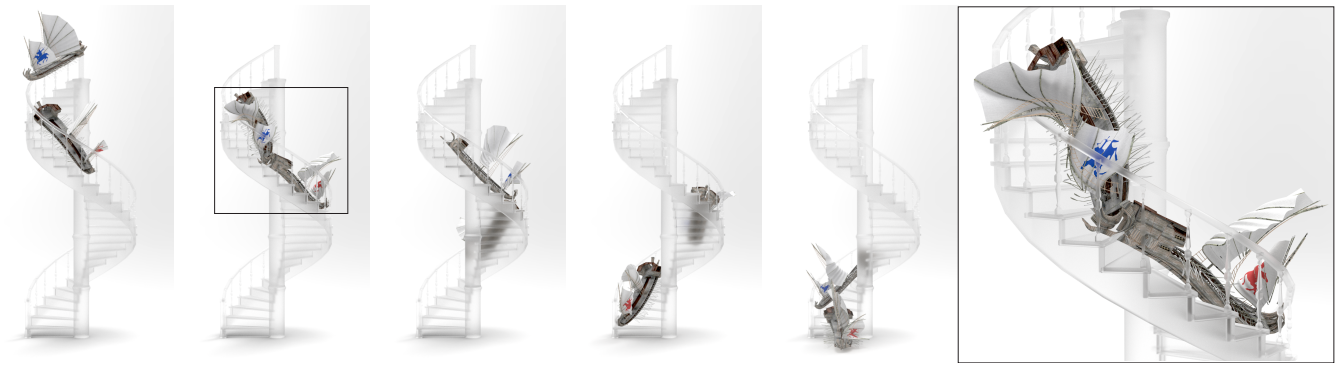

Fig. 1. **Falling barbarian ships.** We propose a new GPU-based algorithm for generic finite element simulation using interior-point methods. Due to the use of barrier functions, interior-point methods are expensive, and the requirement of per-iteration CCD imposes extra challenges for GPU parallelization. Our method is locally second-order leveraging complex-step finite difference to efficiently estimate local Hessian-vector products. We design a complementary coloring and hybrid sweep scheme to fully exploit the throughput of the GPU. Together with a dedicated warm-start process, our method offers speedup of two orders, even with intense contacts and collisions. As a demonstration, the teaser figure shows snapshots of two barbarian ships falling on a spiral stair. There are nearly one million (974K) elements on the ships. The thin paddles at both sides collide with the staircase and the handrails yielding rich and interesting deformations. Under the time step of $\Delta t = 1/100$ sec, our simulation faithfully captures all the details but it is 129× faster than the vanilla CPU IPC [Li et al. 2020]. Indeed, our GPU simulation is faster than the state-of-the-art reduced simulation [Lan et al. 2021]. The simulation remains efficient and robust even after we increase the time step size to $\Delta t = 1/30$ sec.

In this paper, we present a GPU algorithm for finite element hyperelastic simulation. We show that the interior-point method, known to be effective for robust collision resolution, can be coupled with non-Newton procedures and be massively sped up on the GPU. Newton's method has been widely chosen for the interior-point family, which fully solves a linear system at each step. After that, the active set associated with collision/contact constraints is updated. Mimicking this routine using a non-Newton optimization (like gradient descent or ADMM) unfortunately does not deliver expected accelerations. This is because the barrier functions employed in an interior-point method need to be updated at every iteration to strictly confine the search to the feasible region. The associated cost (e.g., per-iteration CCD) quickly overweights the benefit brought by the GPU, and a new parallelism modality is needed. Our algorithm is inspired by the domain decomposition method and designed to move interior-point-related computations to local domains as much as possible. We minimize the size of each domain (i.e., a stencil) by restricting it to a single element, so as to fully exploit the capacity of modern GPUs. The stencil-level results are integrated into a global update using a novel hybrid sweep scheme. Our algorithm is locally second-order offering better convergence. It enables simulation acceleration of up to two orders over its CPU counterpart. We demonstrate the scalability, robustness, efficiency, and quality of our algorithm in a variety of simulation scenarios with complex and detailed collision geometries.

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

Additional Key Words and Phrases: Physics-based simulation, Interior point method, Barrier function, GPU

ACM Reference Format:
Lei Lan, Minchen Li, Chenfanfu Jiang, Huamin Wang, and Yin Yang. 2023. Second-order Stencil Descent for Interior-point Hyperelasticity. *ACM Trans. Graph.* 42, 4, Article 108 (August 2023), 16 pages. https://doi.org/10.1145/3592104

Authors' addresses: Lei Lan, The University of Utah, USA, lanlei.virhum@gmail.com; Minchen Li, UCLA, USA, minchernl@gmail.com; Chenfanfu Jiang, UCLA, USA, Chenfanfu.Jiang@gmail.com; Huamin Wang, Style3D Research, China, wanghmin@gmail.com; Yin Yang, The University of Utah & Style3D Research, USA, yin.yang@utah.edu.

0730-0301/2023/8-ART108 $15.00
https://doi.org/10.1145/3592104

1 INTRODUCTION

Newton's method has been a popular choice [Baraff and Witkin 1998] for solving the variational form [Kane et al. 2000; Martin et al. 2011] associated with various deformable models. Recently, many contributions suggest that more efficient simulation is possible using non-Newton or quasi-Newton solvers [Hecht et al. 2012; Li et al.

2019; Liu et al. 2017; Narain et al. 2016; Wang and Yang 2016; Wang et al. 2020]. The key insight is to leverage the fact that a deformation process is normally smooth and continuous. A full Newton solve is less economical than slower-converging but more parallelizable iterative methods. This observation also endorses the use of smaller time steps [Macklin et al. 2019].

This paradigm fails with the presence of collisions and contacts. Under such circumstances, the assumption of continuous deformation is severely violated by abrupt and acute interactions among 3D models. In theory, a collision is also smooth [Baraff and Witkin 1992]. It occurs when two objects come into contact under a positive relative velocity (i.e., they are moving toward each other). After colliding, the kinetic energy transforms into the elastic potential. The latter accumulates and releases eventually to generate a repulsive force pushing models apart. This physical change is well understood and can be directly observed with a high-speed camera [Labous et al. 1997]. However, it appears *invisible* under a common time step size of tenths or hundredths of a second, and the conversion between kinematic and elastic energies is hidden behind inequality constraints. Robust treatment of those constraints under insufficiently sampled time discretization becomes a major challenge.

The interior-point method seems to be a promising solution. As demonstrated in recent contributions [Li et al. 2020, 2021b] the inequality constraints, due to contact and collision, can be well handled using barrier functions, which yield increasingly stronger penalties when constraints are about to become violated. This method is named incremental potential contact or IPC. The robustness of IPC comes from its line search mechanism, which must be certified by a continuous collision detection (CCD) routine. The CCD ensures a position update is always within the *feasible region* i.e., where the inequality constraints are strictly satisfied, and the barrier functions are well-defined. That said, any displacement update must be accompanied by a CCD as long as barriers are part of the optimization. Such binding to CCD raises the hidden cost for each iteration and imposes significant difficulties in improving the scalability and efficiency of the interior-point family.

In this paper, we propose a GPU solution for efficient and scalable barrier-enabled simulations. Due to the dependence on CCD, we divert our focus from fast per-iteration computation i.e., as in most existing GPU solvers, to improved convergence. This objective is achieved by the coordination of 1) a second-order parallel solver; 2) lightweight local CCD; 3) complementary coloring with a hybrid update scheme; 4) and a better warm start.

Our method is locally second-order. We find local curvature information highly effective in relaxing the nonlinearity induced by barriers. When a collision pair (i.e., a vertex-triangle or an edge-edge pair) stays nearby, a regular CCD often yields a close-to-zero time of impact (TOI). Such a small TOI "freezes" the whole simulation as displacement updates of other vertices are also truncated. We name this issue *TOI locking* as more iterations must be followed until this colliding pair is fully resolved. Therefore, vanilla IPC is inefficient for collision-intensive scenes (despite its robustness). We contrive a local CCD scheme, which allows those nearby pairs to be solved locally while other freedoms can keep deforming.

The scalability and efficiency are enabled by parallel local solves over a subset of unknown degrees of freedom (DOFs). We refer to each subset as a *stencil*, which corresponds to a tetrahedron on the model or a vertex-triangle or an edge-edge collision pair. The updates of stencils become independent if they are not connected on the mesh or coupled by a barrier function. Normally, such a Gauss-Seidel-like scheme requires coloring irrelevant stencils based on their connectivity [Fratarcangeli and Pellacini 2015; Fratarcangeli et al. 2016; Ton-That et al. 2022]. As collision pairs vary dynamically, the coloring also needs to be recomputed at each iteration. We propose a complementary coloring method and mixed update strategy, which avoid recoloring for different collision configurations and improve the convergence. With a dedicated warm-start step, our algorithm runs substantially faster than its CPU counterparts.

In addition, our method does not rely on simplification of the material models (as opposed to projective dynamics [Bouaziz et al. 2014; Lan et al. 2022b] or position-based dynamics [Macklin et al. 2016]) and can deal with any hyperelastic material. It is less sensitive to the stiffness of the model or the time step size, thanks to local second-order relaxations. Our iteration count is comparable to Newton's method, but our approach gives speedups of two orders in general. In fact, it is able to match the state-of-the-art subspace simulation performance without using reduced-order models.

2 RELATED WORK

Deformable body simulation has been an active graphics research topic since the 1980s [Terzopoulos and Fleischer 1988; Terzopoulos et al. 1987, 1988]. The goal of deformable simulation is to replicate real-world material behaviors digitally, and it has been an indispensable ingredient in a wide range of applications like surgical training [Meier et al. 2005], fabrication [Vanek et al. 2014], robotics [Umedachi et al. 2013], AR/VR [Popescu et al. 1999], digital fashion [Choi and Ko 2005a; Wang 2018] etc.

Due to the numerical stiffness of deformable models, implicit time integration methods like backward Euler [Baraff and Witkin 1998] or Newmark [Hughes 2012] are mostly chosen. Doing so improves the stability of the integration but leaves a global (often sparse) linear system to solve. This computation is expensive and stands as the major bottleneck of the simulation pipeline. A natural thought is to avoid a full linear solve in classic Newton's method. Following this idea, Hecht and colleagues [2012] proposed a lagged factorization scheme that reuses existing Cholesky factorization to save the computation. Multi-resolution [Capell et al. 2002b; Grinspun et al. 2002] and multigrid solvers project fine-grid residual errors onto a coarser grid, on which linear or nonlinear iterations are more effective [Bolz et al. 2003; Tamstorf et al. 2015; Wang et al. 2020; Xian et al. 2019; Zhu et al. 2010]. Quasi-Newton methods use Hessian approximations, instead of the exact Hessian, to estimate a good search direction [Li et al. 2019; Liu et al. 2017; Wang et al. 2020].

Idealizations of the elasticity model also lead to several important simulation techniques. A classic example would be stiffness warping [Müller et al. 2002], which reuses rest-shape stiffness matrix for large rotational deformation. This method can also be combined with modal analysis to enable real-time simulations [Choi and Ko 2005b]. Chao and colleagues [2010] designed a simplified material model measuring the distance of linear deformation and rotation. This concept is similar to the shape matching algorithm [Müller et al.

2005], where the deformation energy is defined based on the nearest rigid body transformation. Position-based dynamics (PBD) [Macklin et al. 2016; Müller et al. 2007] regards the elastic energy as a set of compliant constraints and uses steepest descent to update the vertex positions. This method is later generalized to handle fluid [Macklin and Müller 2013] and rigid bodies [Müller et al. 2020]. Similarly, projective dynamics (PD) separates the constraint projection and the distance measure into local and global steps [Bouaziz et al. 2014]. The key benefit of PBD and PD is the (partial) decouple of DOFs in different constraints. As a result, both methods can be well parallelized on the GPU [Fratarcangeli et al. 2016, 2018; Wang 2015]. Those existing fast simulation algorithms rely on the assumption that the deformation occurs smoothly along the time, and a full second-order Newton iteration could be replaced with multiple but less costly first-order ones [Wang 2015].

Model reduction is another popular acceleration technique a.k.a. the subspace method or reduced-order models, which creates a subspace representation of fullspace DOFs. Modal analysis [Choi and Ko 2005b; Hauser et al. 2003; Pentland and Williams 1989] and its first-order derivatives [Barbič and James 2005] are often considered the most effective method for subspace construction. Displacements from recent fullspace simulations can also be utilized [Kim and James 2009]. Prior art also coarsens the geometric representation to prescribe the dynamics of a fine model. For instance, Capell and colleagues [2002a] deformed an elastic body using an embedded skeleton; Gilles and colleagues [2011] used 6-DOF rigid frames to drive the deformable simulation; Faure and colleagues [2011] used scattered handles to model nonlinear dynamics; Lan and colleagues [2020; 2021] exploited the medial axis transform to build the mesh skeleton; Martin and colleagues [2010] used sparsely-distributed integration points named elastons to model the nonlinear dynamics of rods, shells, and solids uniformly. Since the number of simulation DOFs does not depend on the resolution of the model, orders-of-magnitude speedups are not uncommon with reduced simulation. On the downside, the accuracy compromise and the loss of simulation details are inevitable – after all, it compresses a high-dimension simulation into a low-dimension space.

The presence of collisions and contacts imposes an extra layer of difficulty for deformable simulation. A collision occurs in a short period of time leading to rapid velocity/position changes. Collisions have been modeled by impulses in early works [Baraff 1989; Mirtich and Canny 1995; Moore and Wilhelms 1988; Weinstein et al. 2006]. Doing so stiffens the simulation, leading to undesired artifacts and failure of the system to converge when the non-penetration constraint must be strictly enforced [Cline and Pai 2003]. Switching to complementarity programming does not resolve this issue. As it is an NP-hard problem, we often do not have the luxury to run the complementarity search to the end for an optimal solution [Anitescu and Potra 1997; Erleben 2007; Kaufman et al. 2005]. Consequently, the simulation remains inconsistent and unstable. An inexpensive alternative is the penalty method [Cundall and Strack 1979; Terzopoulos et al. 1987; Teschner et al. 2005]. Instead of using inequality constraints, a penalty method chooses a spring-like repulsion mechanism based on the penetration depth between two objects [Drumwright 2007; Fisher and Lin 2001; Hasegawa et al. 2004; Wu et al. 2020]. While straightforward, the penalty method fails for fast-moving models or simulations under bigger time steps and often requires tedious tuning of stiffness parameters per scene. Its stability can be enhanced using implicit formulations coupled with CCD [Tang et al. 2012; Xu et al. 2014]. Nevertheless, intersections among models still can and will result.

The interior-point method [Mehrotra 1992] incorporates barrier functions to approximate inequality constraints induced by collisions and contacts. As the name suggests, a barrier function is designed as a nonlinear penalty yielding increasingly stronger repulsion when models are moving closer to each other. Its feasibility and robustness have been validated by Li and colleagues [2020], where they name this barrier-based collision resolution IPC. IPC offers two immediate advantages: 1) Unlike using impulses, IPC smooths the problem formulation with controllable accuracy; 2) IPC enables the algorithmic guarantee that the inequality constraints are always satisfied so the resulting simulation is free of interpenetration. This method has then been successfully employed for reduced simulation [Lan et al. 2021], co-dimensional simulation [Li et al. 2021b], rigid body simulation [Ferguson et al. 2021; Lan et al. 2022a], embedded FEM [Choo et al. 2021], FEM-MPM coupling [Li et al. 2021a], and geometric modeling [Fang et al. 2021].

The CCD-based line search plays a key role in the IPC framework – it ensures that any displacement update is confined within the domain of the barrier function. This also suggests existing GPU algorithms, which trade a costly Newton step for multiple inexact but inexpensive iterations [Narain et al. 2016; Wang and Yang 2016; Wu et al. 2022], become hardly practical since performing culling and CCD at each iteration is prohibitive. Lan and colleagues [2022b] alleviated this difficulty by casting the barrier as a positional constraint in the framework of PD. Unfortunately, for more generic simulations, one has to evaluate the barrier to determine if the current search direction is descending. Without per-iteration CCD, the barrier functions become undefined, and the simulation then fails.

The existence of barriers is a double-edged blade. On the one hand, it converts inequality-constrained simulation to an unconstrained one and allows highly robust resolutions of collisions and contacts. On the other hand, it ties any iterative solvers with CCD, ruling out most parallel strategies currently available. In this paper, we present a non-Newton and parallel solver, dedicated to barrier-in-the-loop deformable simulations. Our algorithm is closely related to the domain decomposition method [Farhat et al. 2000] by breaking the original model into many small domains. While this idea is not new in graphics and has been employed with model reduction for per-domain subspace customization [Barbič and Zhao 2011; Kim and James 2012; Wu et al. 2015; Yang et al. 2013], we show that it also leads to scalable and parallelizable nonlinear programming. Our method is also relevant to the coordinate descent method [Naitsat et al. 2020; Wright 2015], which is a simple divide-and-conquer solution for large-scale optimizations. We perform a local second-order Newton-like optimization over blocks of coordinates and align each of such blocks with an element on the model.

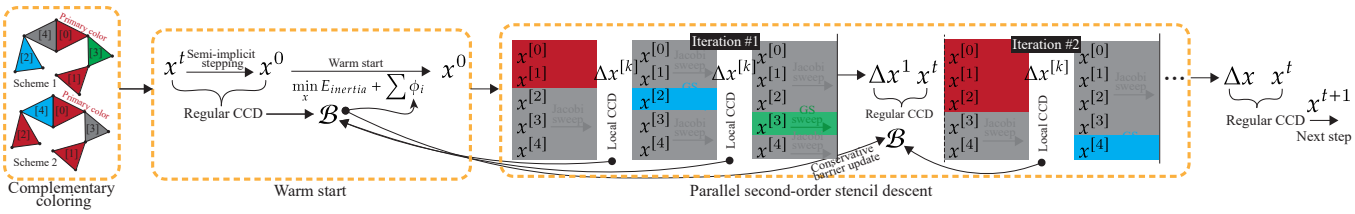

Fig. 2. **Algorithm overview.** We present a local second-order GPU solver to efficiently simulate barrier-in-the-loop elastic models. An input model will be colored based on the connectivity of stencils with several complementary schemes (§ 5). At each time step, a warm start is carried out first building a global collision list with an altered per-stencil barrier formulation (§ 6). The core relaxation runs in parallel using a hybrid sweep strategy based on computed color schemes (§ 4), where we use local CCD during the iterations as much as possible.

3 ALGORITHM OVERVIEW & PRINCIPAL DESIGN

Before jumping into the details of our method, we first give a brief review of the interior-point method and explain why a different algorithm is needed for barrier-in-the-loop elastic simulation.

Assume the implicit Euler is used, the elastic simulation is formulated as the variational optimization of:

$$x^{t+1} = \arg\min_x \frac{1}{2}(x-\widetilde{x})^\top \mathsf{M}(x-\widetilde{x}) + h^2\Psi(x) \text{ s.t. } h_i(x) \geq 0. \quad (1)$$

Here, x^{t+1} represents the unknown positional DOFs of all the vertices on the model i.e., a tetrahedral mesh. $\widetilde{x} = x^t + hv^t + h^2\mathsf{M}^{-1}f_{ext}$ is a known vector depending on the position x^t and velocity v^t from the previous time step as well as the external force f_{ext}. h is the time step size, and M is the mass matrix. The first term $E_{inertia} = \frac{1}{2}(x-\widetilde{x})^\top\mathsf{M}(x-\widetilde{x})$ in Eq. (1) is sometimes also referred to as the inertia potential. $\Psi(x)$ denotes the hyperelastic energy measuring the "magnitude" of the deformation. $h_i(x) \geq 0$ form a set of C inequality constraints enforcing the simulation to be free of inter- and intra-model intersections. Eq. (1) is mathematically equivalent to an unconstrained optimization using *indicator functions* $I_\Omega(x)$, which evaluates $+\infty$ if $x \notin \Omega$ and 0 otherwise:

$$\min_x \frac{1}{2}(x-\widetilde{x})^\top\mathsf{M}(x-\widetilde{x}) + h^2\Psi(x) + \sum_{i=0}^{C} I_{\{h_i\geq 0\}}(x). \quad (2)$$

IPC [Li et al. 2020] is a primal implementation of the interior-point method, which approximates $h_i(x) \geq 0$ with logarithmic barrier functions:

$$\sum_{i=1}^{C} I_{\{h_i\geq 0\}} \approx \kappa \sum_{i=1}^{C} \phi_i(x), \quad (3)$$

where each barrier ϕ_i is defined as:

$$\phi_i(d_i,\hat{d}) = \begin{cases} -(d_i-\hat{d})^2 \log\left(\frac{d_i}{\hat{d}}\right), & 0 < d_i < \hat{d} \\ 0, & d_i \geq \hat{d} \end{cases}. \quad (4)$$

$\hat{d}$ is a global hyper-parameter prescribing the accuracy of the approximate in Eq. (3). Intuitively, it allows ϕ_i to be "active" if the closest distance between a collision pair (i.e., d_i) is smaller than $\hat{d}$. Similar to $I_{\{h_i\geq 0\}}$, ϕ_i approaches to $+\infty$ as d_i approaches to 0, scaled by κ. IPC then aims to find the optimal solution to:

$$\min_x E(x),\ E = \frac{1}{2}(x-\widetilde{x})^\top\mathsf{M}(x-\widetilde{x}) + h^2\Psi(x) + \kappa\sum_{i=0}^{C}\phi_i(x). \quad (5)$$

As nonlinear programming iteratively seeks a better solution to Eq. (5), one needs to ensure that each displacement update Δx is descent and lowers the target function $E(x)$. Note that $\phi_i(x)$ is only defined for $d_i > 0$ per Eq. (4). If Δx takes any $\phi_i(x)$ out of its domain, there is no way for us to validate whether Δx is a legit improvement. Thus, an interior-point algorithm like IPC always equips each iteration with a CCD to ensure $\phi_i(x)$ are well-defined. Being taxed by CCD, slow-converging methods like gradient descent [Wang and Yang 2016] are no longer an option.

Parallel algorithms break the computation into smaller subsystems, and the convergence-efficiency trade-off is embodied via different choices of 1) the size of a subsystem 2); how each subsystem is solved, and 3) how subsystems are coupled/integrated. In general, smaller subsystems, lower-order methods, and less information sharing improve efficiency, but the simulation will need much more iterations for stiffer problems. By contrast, we choose to solve a bigger subsystem using a second-order method and a much stronger subsystem coupling mechanism.

Fig. 2 sketches an overview of our method. The core computation of our pipeline is a local second-order relaxation scheme namely stencil descent (§ 4). The use of local curvature information greatly improves the convergence for barrier-enabled problems. Unfortunately, this method alone is unable to deliver the desired performance. We design a simple and effective coloring method that can be pre-computed and is robust against different collision patterns (§ 5). This coloring method, combined with a hybrid update scheme, allows the simulation to converge at a similar pace as Newton's method. The performance of our solver is further enhanced by a warm-start process using an altered barrier formulation (§ 6).

4 SECOND-ORDER STENCIL DESCENT

Coordinate descent is a well-known optimization algorithm [Wright 2015] that minimizes a multivariable function e.g., $E(x)$ in Eq. (5) by optimizing one DOF (coordinate) at a time successively. The convergence of coordinate descent can be easily confirmed, as each local iteration always lowers the target function [Zheng et al. 2000].

We generalize this concept by finding a descent direction of a group of DOFs at once. Specifically, each group houses 12 DOFs corresponding to 1) a tetrahedron element on the model or 2) a colliding vertex-triangle or edge-edge pair with an activated barrier namely, an *elasticity stencil* or a *barrier stencil*. The reason behind this choice is multifaceted. First, a tetrahedron forms a minimal 3D simplex,

which uniquely determines its own rigid body transformation. As a result, the residual in the null space can be effectively suppressed locally, which better improves the global convergence than smaller subsystems e.g., per vertex. Being a simplex, tetrahedrons do not have isolated DOFs (except for the ones at the boundary corners) meaning all the DOFs are shared by their neighbors. Therefore, choosing the tetrahedron as the basic coordinate unit strengthens the coupling among stencils and enables faster strain propagation. Finally, a 12-dimension stencil remains compact and can still be efficiently solved using second-order methods even with limited hardware resources at each CUDA thread.

4.1 Projection-free Stencil-wise Newton-CG

In our algorithm, the k-th stencil solves an optimization of Eq. (5) only w.r.t. its local DOFs, $x^{[k]} \in \mathbb{R}^{12}$:

$$\min_{x^{[k]}} E(x) \equiv \min_{x^{[k]}} E^{[k]}. \tag{6}$$

Here the superscript $^{[k]}$ denotes the index of the stencil. Note that $E^{[k]}$ accumulates not only the elasticity or the barrier of the stencil itself but also energies shared by its neighbors. Any vertex-sharing, edge-sharing, and face-sharing neighboring elements contribute to $E^{[k]}$, and we have $\sum E^{[k]} \geq E(x)$.

We use standard Newton's method to solve Eq. (6) at each stencil, and compute a *tentative* local update $\Delta x^{[k]}$ as:

$$\Delta x^{[k]} = -\left(\mathrm{H}^{[k]}\right)^{-1} g^{[k]}, \tag{7}$$

where $\mathrm{H}^{[k]}$ and $g^{[k]}$ are local Hessian and gradient. Again, both $\mathrm{H}^{[k]}$ and $g^{[k]}$ are different from the element Hessian/gradient, as they also accommodate the first- and second-order derivatives of neighbor stencils.

Solving the 12-by-12 linear system of Eq. (7) causes practical obstacles. The analytical formulation is too complicated, if exists, to be hard-coded while a local Gaussian elimination is logically overwhelming for one CUDA thread. To this end, we choose to use CG (conjugate gradient method) for our per-stencil relaxation (Alg. 1). CG returns the exact solution of Eq. (7) after 12 iterations because the error at each of 12 conjugated directions will be right eliminated at the corresponding iteration [Shewchuk et al. 1994]. Indeed, CG was invented as a direct solver originally. The stencil-level CG also frees us from an explicit Hessian assembly because CG iterations only need Hessian-vector product (i.e., $\mathrm{H}^{[k]}p_j$ at lines 3 and 5 in Alg. 1), which is the *directional derivative* of $g^{[k]}$: $\mathrm{H}^{[k]}p_j = \nabla_{p_j} g^{[k]}$ [Shen et al. 2021; Yang et al. 2015].

Large deformations of nonlinear materials often yield a Hessian that is not positive semi-definite (PSD), and the resulting $\Delta x^{[k]}$ becomes errant. To restore the numerical stability, PSD-projected Hessian is often used [Teran et al. 2005] a.k.a. projected Newton. We note that CG can be used to solve local stencil robustly without PSD projection in a Hessian-free manner.

To see the reason behind this, let us re-visit Newton's method, which Taylor expands $E^{[k]}$ around $x^{[k]}$:

$$E^{[k]}(x^{[k]} + \Delta x^{[k]}) = E^{[k]}(x^{[k]}) + Q(\Delta x^{[k]}) + O(\|\Delta x^{[k]}\|^3). \tag{8}$$

Here, $Q(\Delta x^{[k]}) = \frac{1}{2}\Delta x^{[k]\top}\mathrm{H}^{[k]}\Delta x^{[k]} + \Delta x^{[k]\top} g^{[k]}$ is a quadratic form of $\Delta x^{[k]}$. Newton's method ignores the third-order error and seeks the minimizer of $Q(\Delta x^{[k]})$, while CG happens to be a dedicated algorithm for minimizing $Q(\Delta x^{[k]})$. Such coherence also makes the Newton-CG family one of the most popular numerical solutions for large-scale optimizations [Zhao et al. 2010].

ALGORITHM 1: Per-stencil CG iteration.

1: $r_0 \leftarrow -g^{[k]} - \mathrm{H}^{[k]}\Delta x_0^{[k]}$, $p_0 \leftarrow r_0$, $j \leftarrow 0$;
2: **while** $j < 12$ **do**
3: $\quad \alpha_j \leftarrow \begin{cases} \frac{1}{\sigma}, & p_j^\top \mathrm{H}^{[k]} p_j \leq 0 \\ \frac{r_j^\top r_j}{p_j^\top \mathrm{H}^{[k]} p_j}, & \text{otherwise} \end{cases}$;
4: $\quad \Delta x_{j+1}^{[k]} \leftarrow \Delta x_j^{[k]} + \alpha_j p_j$;
5: $\quad r_{j+1} \leftarrow r_j - \alpha_j \mathrm{H}^{[k]} p_j$;
6: $\quad \beta_j \leftarrow \frac{r_{j+1}^\top r_{j+1}}{r_j^\top r_j}$;
7: $\quad p_{j+1} \leftarrow r_{j+1} + \beta_j p_j$;
8: $\quad j \leftarrow j + 1$;
9: **end**

If $\mathrm{H}^{[k]}$ is PSD, every CG iteration will lower $Q(\Delta x^{[k]})$[1]. When this is not the case, $\mathrm{H}^{[k]}$ then has non-positive eigenvalues, and searching along the corresponding eigenvectors may increase the value of $Q(\Delta x^{[k]})$. This suggests a bad direction that one should avoid. Geometrically, a negative eigenvalue corresponds to some direction along which the curvature of the Hessian is concave. Projected Newton [Teran et al. 2005] directly "flattens" concave regions to make it mildly convex. In CG, such concavity of a non-PSD Hessian is reflected by $p^\top \mathrm{H}^{[k]} p$ (line 3). $p^\top \mathrm{H}^{[k]} p \leq 0$ implies p largely aligns with negative eigenvectors, and the trajectory along p is concave. Following the same strategy of projected Newton, we manually alter local negative curvature by setting $p^\top \mathrm{H}^{[k]} p \leftarrow \sigma \|r_j\|^2$ i.e., a small positive quantity relative to the current residual norm (line 3 of Alg. 1). Doing so allows CG to travel out of this concave region and keep reducing residual in other conjugate directions. Alternatively, one can simply quit the current CG iteration. In practice, concave searches are rare, and few early terminations do not impact the overall convergence of our method.

4.2 Local CCD & Inversion Search

The computation of $\Delta x^{[k]}$ is local and agnostic on potential geometric or topological conflicts. Therefore, additional sanity checks are needed namely, local CCD search and local inversion search.

The so-called local CCD performs a lightweight CCD over $\Delta x^{[k]}$. It does not exhaustively search nearby vertex-triangle and edge-edge pairs as in a regular CCD routine. Instead, *local CCD just makes sure that existing collision pairs, identified by the most recent regular CCD, do not generate intersections.* Local CCD is quit fast without any culling or neighborhood search. We use the fast polynomial solver

[1] Minimizing $Q(\Delta x^{[k]})$ does not necessarily mean Eq. (6) is improved due to the existence of $O(\|\Delta x^{[k]}\|^3)$. Therefore, a line search is still needed even for a PSD Hessian.

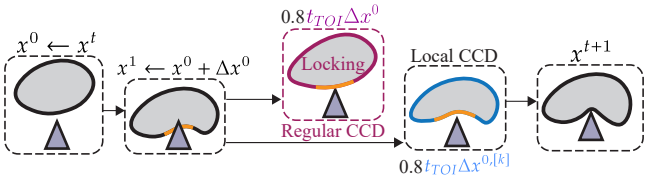

Fig. 3. **Regular and local CCD.** Local CCD is a simple and effective treatment for local displacement updates. The grey-colored deformable body hits the tip of the collider. If the TOI at the colliding element (in orange) is small, regular CCD filtering down scales the global update Δx by $0.8t_{TOI}$, which prevents the deformation of the whole model in following iterations. Local CCD only adjusts the displacement of the colliding stencil, and other stencils can keep updating their own $\Delta x^{[k]}$.

Fig. 4. **Hybrid sweep.** Hybrid sweep follows the Gauss-Seidel style update but fills idle GPU threads with Jacobi updates. Therefore, all stencils will be updated at each sweep. This simple strategy improves the convergence without noticeable GPU latency.

proposed by Yuksel [2022] to compute the local TOI. While the cost of local CCD is subtle, it is a vital treatment in our framework offering three important guarantees:

(1) Local CCD ensures that all the existing barrier functions are well-defined so that per-stencil line search is meaningful.
(2) Local CCD retains the formulation of current target function (Eq. (5)) by not adding new barriers or removing old ones.
(3) Local CCD uses local TOI to adjust $\Delta x^{[k]}$ of each stencil to avoid TOI locking due to small global TOIs.

The third guarantee is particularly helpful for faster convergence. As illustrated in Fig. 3, as soon as a collision pair yields a close-to-zero TOI, the corresponding global displacement update will be blocked since Δx needs to be scaled by $0.8t_{TOI}$ to keep the barrier well-defined[2] – even though other parts of the models are free of collisions. It appears as the entire model is stiffened, waiting for the relaxation of this specific barrier. The corresponding computations are then wasted. Local CCD mitigates this issue since this small TOI only affects local stencils, and other stencils keep deforming based on their local descent directions.

Local inversion search is for elasticity stencils only, which prevents newly computed $\Delta x^{[k]}$ from generating element inversions. The first pass of the inversion search checks whether the following equation of t has a root between $(0, 1]$:

$$\begin{vmatrix} \left(x_2^{[k]} + t\Delta x_2^{[k]} - x_0^{[k]} - t\Delta x_0^{[k]}\right)^\top \\ \left(x_1^{[k]} + t\Delta x_1^{[k]} - x_0^{[k]} - t\Delta x_0^{[k]}\right)^\top \\ \left(x_3^{[k]} + t\Delta x_3^{[k]} - x_0^{[k]} - t\Delta x_0^{[k]}\right)^\top \end{vmatrix} = 0. \tag{9}$$

Here $x_i^{[k]}$, $i = 0, 1, 2, 3$ are positions of four vertices of the stencil. $\Delta x_i^{[k]}$ are their respective displacements. If $\Delta x^{[k]}$ does not invert stencil $[k]$, the second pass iterates all the incidence elements and checks if $\Delta x_i^{[k]}$ displaces vertices to the negative side of opposite faces in those elements. Let x_s, x_t, and x_l be the three vertices on the opposite face of vertex $x_i^{[k]}$. The inversion search checks whether

$$t = -\frac{[(x_s - x_l) \times (x_t - x_l)] \cdot (x_i^{[k]} - x_l)}{[(x_s - x_l) \times (x_t - x_l)] \cdot \Delta x_i^{[k]}} \tag{10}$$

[2]Here, 0.8 is a parameter of user's choice, which should be slightly smaller than 1 to keep the displacement update intersection-free.

is between $(0, 1]$. If so, we scale $\Delta x^{[k]}$ by $0.8t$. Lastly at this step, after local CCD and inversion check, a local line search ensues. We choose a simplified Wolfe condition and make sure $\Delta x^{[k]}$ lowers $E^{[k]}$ and thus improves E overall.

Like all the existing GPU algorithms [Fratarcangeli et al. 2016, 2018; Wang 2015; Wang and Yang 2016], our speedup comes from the parallelization of local computations at elasticity and barrier stencils. To make the most use of the GPU, we design a novel hybrid sweep strategy that avoids per-iteration graph coloring.

5 PARALLELIZATION

Stencils share DOFs, and porting local $\Delta x^{[k]}$ to the global Δx can only be carried out for disjoint stencils. The parallelization relies on graph coloring [Jensen and Toft 2011], which abstracts the connectivity among stencils as an undirected graph. Stencils in the same color are independent, and their local $\Delta x^{[k]}$ can be copied to global displacement without conflicts. It is desired that each color group houses as many disjoint stencils as possible for maximized parallelism, while an over-dominant color inevitably shrinks other colors and leads to *unbalanced coloring*. Due to the collision, stencil connectivity varies at each iteration, and finding an optimal coloring for non-planar graphs is NP-complete. Therefore, existing methods resort to heuristics to do coloring on the fly [Fratarcangeli et al. 2016, 2018; Ton-That et al. 2023]. We present a pragmatic solution to this challenge, which allows us to pre-compute all the colors before the simulation.

5.1 Hybrid Sweep

For models with hundreds of thousands of elements, heuristic coloring algorithms typically generate a few dozen colors, where each color has up to tens of thousands of stencils. Running stencil descent in parallel, even for the largest color, is way below the GPU capacity. For instance, we color a dragon model (Fig. 13) of 1M elements using Vivace [Fratarcangeli et al. 2016]. It takes 308 ms on average to relax stencils of one color. Meanwhile, completing 12 CG iterations at all the stencils only needs 339 ms.

To make the most out of the GPU, our hybrid sweep consists of a Gauss-Seidel sweep and a Jacobi sweep. They jointly compute $\Delta x^{[k]}$ for all stencils i.e., see Figs. 2 and 4. After local $\Delta x^{[k]}$ are ready (which do not depend on the coloring of stencils), stencil displacements of the current color are updated with the highest priority in a Gauss-Seidel manner followed by Jacobi sweep, which averages displacements of vertices shared by multiple stencils *without modifying the committed Gauss-Seidel update*. Local CCD, inversion,

and line searches are performed at each sweep separately. In other words, a hybrid sweep combines one parallel Gauss-Seidel step for stencils in one color group and one Jacobi iteration for all the other stencils. This strategy feeds unoccupied GPU threads with the Jacobi sweep. Assigning Jacobi sweeps at spare threads is nearly "free", and they quietly improve the objective function in the background. As a result, the hybrid sweep can accelerate the overall convergence over 30% without noticeable latency. Our method should not be confused with simply interleaving Gauss-Seidel and Jacobi iterations as mentioned in [Bender et al. 2017]. We would like to remind that either Gauss-Seidel or Jacobi sweeps refers to how local displacements are updated. Our underlying solver is locally second-order, which converges faster than using Gauss-Seidel or Jacobi as linear solvers.

Hybrid sweep also makes our parallelization resilient to the topological change of the graph as we can keep barrier stencils with Jacobi sweeps (i.e. grey color). This is reasonable because barriers are often localized, and Jacobi-style relaxation is quite effective for isolated barriers. In addition, our pipeline also includes a warm start that pre-processes predicted barrier stencils (see § 6).

5.2 Complementary Coloring

Hybrid sweep is not sensitive to unbalanced coloring since all the stencils are to be updated anyway. A small-size color may be less helpful in vanilla Gauss-Seidel, but the accompanying Jacobi-sweep greatly enhances its efficacy. This property allows a simple and straightforward coloring algorithm, wherein we can focus on maximizing the dominant color group. As described in Alg. 2, our coloring procedure is based on the classic Welsh-Powell method [Welsh and Powell 1967]. We name the first-assigned color the *primal color*, which tends to have more stencils. We always try to color a stencil with the least indexed color available. During this process, new colors are generated to accommodate conflicting stencils. We stop adding more colors after the total number of colors reaches a threshold T and categorize all uncolored stencils into *grey color*. Those grey stencils are updated via Jacobi sweep as they conflict with other colors.

To cancel the bias induced by fixed-order Gauss-Seidel sweeps [Simon 1992], we build multiple color schemes and make them "complementary" to each other. Specifically, we ensure that each elasticity stencil is assigned as the primal color at least once in a scheme so that it will undertake a major Gauss-Seidel sweep during the iteration. We make sure that an elasticity stencil is not in grey color in all the schemes. Such complementarity is enabled by sorting stencils with different metrics. In the first color scheme, stencils are sorted according to their degrees on the connectivity graph. In the following schemes, the metric switches to the sum of color indices a stencil previously received. Note that we have our primary color indexed as 0 and the grey color index as $+\infty$ (lines 7 and 13 in Alg. 2). For high-resolution models with many elasticity stencils, comparison-based sorting could be expensive. We employ counting-based sorting i.e., radix sort [Davis 1992] which has the linear complexity for a small number of colors. Nevertheless, the coloring procedure is before the simulation. Fig. 5 shows two color schemes of a dragon model.

6 WARM START USING CUSTOMIZED BARRIERS

ALGORITHM 2: Complementary coloring.

1: **for** $i = 1$ *to len(L)* **do**
2: | rank($L[i]$) = degree of stencil $L(i)$ // L is the stencil list
3: **end**
4: **repeat**
5: | Sort all stencils in L according to rank($L[i]$);
6: | **for** $i = 1$ *to len(L)* **do**
7: | | $c \leftarrow 0$; // 0 is the primary color
8: | | **repeat**
9: | | | Tentatively color stencil $L[i]$ with color c ;
10: | | | $c \leftarrow c + 1$;
11: | | **until** *No conflict found*;
12: | | **if** $c \geq T$ **then**
13: | | | color($L[i]$) $\leftarrow +\infty$; // $+\infty$ is the grey color
14: | | **end**
15: | | rank($L[i]$) $\leftarrow$ color($L[i]$) + rank($L[i]$);
16: | **end**
17: **until** *All color schemes are generated*;

Fig. 5. **Complementary coloring.** Our hybrid sweep is used with a complementary coloring strategy. The coloring of stencils is pre-computed, and we generate multiple color schemes to make sure each stencil shows up in an independent color group at least in one scheme. Here, we show first four colors of two schemes on a dragon model.

ALGORITHM 3: Inertia-barrier warm start.

1: $x^0 \leftarrow x^t + hv^t + \frac{1}{4}h^2(a^t + \mathsf{M}^{-1}f_{ext})$;
2: CCD(x^t, x^0);
3: Generate barrier list $\mathcal{B}$;
4: $x^0 \leftarrow \arg\min_x \frac{1}{2}(x - \widetilde{x})^\top \mathsf{M}(x - \widetilde{x}) + \sum \phi_i$;

An important ingredient of our pipeline is a warm-start procedure for a better initial guess of x^0. Our warm-start strategy is inspired by time splitting [Wicker and Skamarock 2002] and relaxes the stiffest component of Eq. (5) i.e., the barrier energy in advance. It positions the system to a collision-free state where the majority of barriers are already relaxed. This initialization dampens abrupt and impulse-like barriers induced by fast-moving vertices and therefore facilitates the convergence of descent iterations.

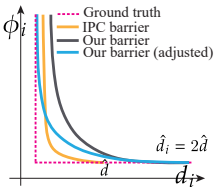

Fig. 6. **Per-stencil barrier**. By modifying barrier formulation, we can early start the relaxation of barrier stencils.

As shown in Alg. 3, we start with a semi-implicit prediction: $x^0 \leftarrow x^t + hv^t + \frac{1}{4}h^2(a^t + \mathsf{M}^{-1}f_{ext})$. Based on the resulting x^0, we compute $\mathcal{B}$, a list of vertex-triangle and edge-edge pairs by performing a regular CCD. We interpret these pairs as a good estimate of active barriers when the simulation arrives at $t = 1$. However, if the distance between a pair of primitives is bigger than $\hat{d}$ at $t = 0$ according to Eq. (4), those barriers remain inactive unless a regular CCD later finds $d_i < \hat{d}$ in a later time. If these barriers will show up in the target function eventually, why not take them into account earlier? To this end, we abandon the setting that $\hat{d}$ is defined globally and unchanged during the simulation but assign a different $\hat{d}_i$ for each ϕ_i at each step. This yields a slightly modified barrier formulation of:

$$\phi_i(d_i, \hat{d}_i) = \begin{cases} -(d_i - \hat{d}_i)^2 \log\left(\frac{d_i}{\hat{d}_i}\right), & 0 < d_i \le \hat{d}_i \\ 0, & d_i > \hat{d}_i \end{cases}, \quad (11)$$

where we set $\hat{d}_i$ for all the collision pairs in $\mathcal{B}$ as the vertex-triangle distance or edge-edge distance at $t = 0$. Eq. (11) then activates all the barriers: They hold a vanished value at $t = 0$ but will participate in the optimization and contribute a non-zero Hessian and gradient to stencils. Compared with the vanilla IPC, doing so is similar to relaxing barriers a few iterations earlier during the optimization and therefore, reduces the total iteration count. We understand that advancing the system with a full elasticity solve provides a better prediction. This strategy has also been explored previously [Bridson et al. 2002]. However, solving the elasticity is significantly more expensive, and the prediction does not substantially differ from a semi-implicit guess in practice.

It is known that KKT's complementary slackness of Eq. (1) is:

$$\lambda_i h_i(x) = 0, \quad (12)$$

where $\lambda_i \ge 0$ is the Lagrange multiplier for each inequality constraint. Replacing $I_{h_i \ge 0}$ with $\phi_i(x)$ essentially relaxes Eq. (12) to:

$$\lambda_i h_i(x) = -\kappa(d_i - \hat{d})^2, \quad (13)$$

as a *perturbed KKT*. From this perspective, one could consider $\kappa(\hat{d} - d_i)^2$ as an indicator of the approximate error of barrier-based collision resolution. Assigning each barrier a different $\hat{d}_i$ may potentially increase the error (when $\hat{d}_i > \hat{d}$), which can be compensated by reducing κ as $\kappa_i \leftarrow \kappa \frac{\hat{d}}{\hat{d}_i}$ (see Fig. 6).

After $\mathcal{B}$ is generated, we further improve x^0 by minimizing the variational problem partially for the inertia and barrier terms:

$$x^0 \leftarrow \arg\min_x E_{inertia}(x) + \sum_{i=1}^{|\mathcal{B}|} \phi_i(x). \quad (14)$$

Similar to barrier stencils, we use parallel Jacobi sweep with local Newton-CG solves so the coloring is not needed for a warm start. Collision pairs are often scattered over the boundary of the model, and $E_{inertia}$ does not strongly couple DOFs. Indeed, $E_{inertia}$ is more like a mass-weighted regularization for barriers $\phi_i(x)$ optimization since barriers are invariant under rigid body motions. Without elasticity Ψ, this problem can be solved with fewer than 10 iterations in most cases. Local CCD and inversion searches are then followed to make sure $E(x^0)$ is well-defined. The computed x^0 is collision-free.

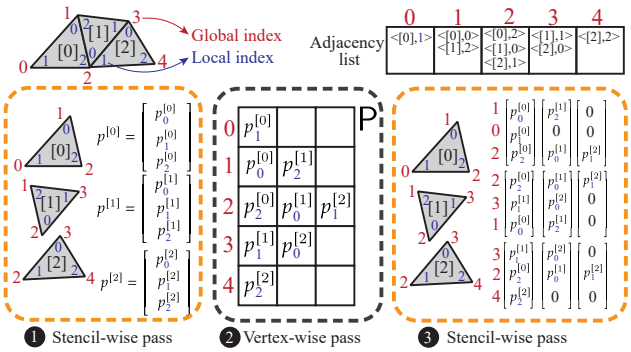

Fig. 7. **Parallel Hessian-vector.** Hessian-vector product at each iteration is computed by taking multiple GPU passes alternating stencils and vertices. This design aims to avoid redundant computations and lows the peak memory demand.

It keeps all the barriers in a "lukewarm" state: They are activated since the distances between primitive pairs at $t = 0$ are more or less shortened by the inertia movement. Meanwhile, as those activated barriers have been processed by stencil descent for a few rounds, strong and sharp penalty is smoothed. The system becomes more regularized and friendly to the follow-up elasticity-aware iterations. In our experiment, we note that this warm-start step could reduce the total iterations by up to 80%.

7 IMPLEMENTATION DETAILS

Being a GPU algorithm, dedicated engineering is essential to deliver the desired performance. In this section, we elaborate on some noteworthy details of the implementation. The common rule of thumb is to understand whether a computation is memory-bounded or computation-bounded. We split lengthy and complicated computations into multiple passes and properly use the shared memory whenever possible.

7.1 Parallel Hessian-vector Product via CSFD

The most dominant computation is the Hessian-vector evaluation. One local iteration yields a 12-dimension displacement update meaning we need to compute, for each vertex, multiple directional force gradients (i.e., $\nabla_{p_i} g^{[k]}$) w.r.t. multiple stencils sharing this vertex. Parallelizing at each vertex naïvely is not efficient because of duplicated stencil visits: A vertex needs to access all the adjacent stencils to compute the respective local directional force gradient.

As shown in Fig. 7, we maintain an auxiliary 2D array P, whose elements are initially set as zero. P has $3N$ rows corresponding to x, y, z components of per-vertex direction, $p_i^{[k]} \in \mathbb{R}^3$, and N is the total number of vertices. The number of columns of P is set as the maximum degree of a vertex i.e., the greatest number of elements shared by a vertex. Our parallelism is achieved with several passes. The first pass is the stencil-wise CG iteration (Alg. 1), which assigns each stencil a local search direction $p^{[k]} \in \mathbb{R}^{12}$. After an iteration is completed (i.e., all stencils are relaxed), we insert P in parallel based on the resultant $p^{[k]}$. In this pass, each vertex iterates its adjacency list, which contains the references of its local copies at each stencil. The third pass computes the Hessian-vector

Table 1. **Time statistics and breakdown.** We give detailed time statistics of experiments reported in the paper. # **Ele.**, # **DOF**, and # **Face** are the total numbers of elements, DOFs, and surface triangles in the simulation. Those are direct metrics indicating the complexities of parallel solve, direct solve, and collision processing. Grey colored face count in the column of # **Face** is the number of triangles on colliders in the scene. # **Color** gives the total number of colors in each test. Δt is the time step size. The columns **Warm start** reports the average iteration counts first and then give the computation time for the warm start. **Stencil descent**, **Local CCD** and **regular CCD** are the total computation time for stencil descent solve, processing local CCD and regular CCD. **Misc.** is the timing for other uncategorized computations such as the gradient evaluation and initialization of auxiliary data structure, etc. # **Iter.** is the average number of iterations for one time step, and **Total** is the total per-step computation time. All the timing is in seconds.

Scene	# Ele.	# DOF	# Face	# Color	Δt	Warm start	Stencil descent	Local line search	Local CCD	Regular CCD	Misc.	# Iter.	Total
Fig. 1	975K	804K	393K + 58K	30	1/100	8 | 0.68	7.6	1.07	1.92	0.86	6.17	102	17.6 (129×)
Fig. 13	1.0M	585K	100K + 1.9K	15	1/100	3 | 0.09	3.9	0.52	0.38	0.3	1.56	16	6.7 (58×)
Fig. 14	487K	402K	197K + 58K	15	1/100	6 | 0.42	6.7	0.66	1.06	0.48	3.49	91	12.5 (91×)
Fig. 15	955K	854K	433K + 25K	15	1/150	26 | 4.0	47.8	27.2	12.3	10.5	19.1	422	116.8 (58×)
Fig. 16	446K	517K	236K + 37K	15	1/100	4 | 0.03	2.1	0.12	0.32	0.66	1.61	27	4.8 (72×)
Fig. 17	3.1M	2.3M	808K + 11K	45	1/100	17 | 6.4	126.8	54.7	36.5	16.1	33.5	321	267.6 (~ 500×)
Fig. 18	1.7M	1.0M	231K + 271K	45	1/100	13 | 0.89	9.2	2.1	1.82	1.8	4.86	67	19.7 (122×)
Fig. 19	342K	387K	129K + 9K	10	1/100	3 | 0.23	4.6	0.31	0.84	0.23	1.36	44	5.1 (25×)
Fig. 21	147K	140K	69K + 10	15	1/100	7 | 0.2	0.43	0.05	0.06	0.066	0.33	17	0.93 (35×)

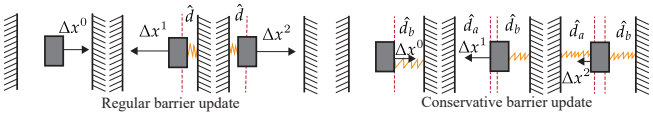

Fig. 8. **Oscillating collision.** Simulation of complex shapes may produce oscillating collisions as vertices could bounce around multiple barriers (left). Conservative barrier update eases this issue. It keeps recent barrier visible during the optimization so that they can be effectively smoothed (right).

products for each column of P that was just filled. This computation spans across stencils to avoid redundancy induced by vertex-based parallelization.

In other words, our strategy re-distributes stencil-based $p^{[k]}$ into multiple vertex-based directions, and the computation of directional force gradient can be unfolded back at stencils. The stencil forms the basic unit for calculating the stress tensor, which is the most costly step and should not be excessively performed. The Hessian-vector product is formulated as: $\mathsf{H}^k p^{[k]} = \nabla_{p^{[k]}} \left(\frac{\partial E^k}{\partial \mathsf{F}^k} \right) : \frac{\partial \mathsf{F}^k}{\partial x^{[k]}}$ for elasticity stencils and $\mathsf{H}^k p^{[k]} = \frac{\partial \phi_i}{\partial d_i} \cdot \nabla_{p^{[k]}} \left(\frac{\partial d_i}{\partial x^{[k]}} \right)$ for barrier stencils. Here H^k denotes the element/barrier stiffness matrix, which should not be confused with $\mathsf{H}^{[k]}$ as the latter also includes the Hessian blocks from adjacent stencils. F^k is the deformation gradient. The directional derivative $\nabla_{p^{[k]}} \partial E^k / \partial \mathsf{F}^k$ and $\nabla_{p^{[k]}} \partial d_i / \partial x^{[k]}$ can be analytically obtained using stress and distance differential [Sifakis and Barbic 2012]. As a more convenient implementation, we used accelerated complex-step finite different (CSFD) [Luo et al. 2019] to compute the directional derivative.

7.2 Conservative Barrier Update

High-resolution and geometry-complex models often have *oscillating* collisions that could take many iterations to be smoothed out. One toy example is visualized in Fig. 8, where a box travels between two colliders. When the box moves close enough to the right wall, the barrier activates (as an orange spring in the figure), which pushes the box to the left at the next iteration. The collider on the left then triggers similar dynamics for the box, keeping it bouncing back and forth between two strong barriers. Such oscillating collisions are particularly perilous in vanilla IPC as it often leads to many close-to-zero TOIs, which in turn freeze global displacement updates (e.g., see Fig. 3).

Conservative barrier update does not rebuild $\mathcal{B}$ immediately after each regular CCD. Instead, we insert novel confirmed barriers without removing old barriers. Keeping old barriers in $\mathcal{B}$ makes them visible to the local CCD so that they can be timely triggered in local updates. As shown in Fig. 8, with the conservative barrier update, the barriers of both colliders are activated, which quickly dampens the oscillation. Note that conservative update should not raise accuracy concerns because barriers have vanished value when $d_i > \hat{d}_i$.

We use hash functions to fast check if a collision pair is already in $\mathcal{B}$. In general, we build two one-dimension hashing tables: one for vertex-triangle pairs and one for edge-edge pairs. Assume that a vertex-triangle collision pair is identified in a regular CCD for the m-th vertex and n-th triangle. The hash function returns:

$$H(m, n) = (m \bmod 3007) \cdot 3007 + (n \bmod 3007). \tag{15}$$

We then check if the hash table cell at $H(m, n)$ is occupied. If so, we double-check the vertex and triangle indices to confirm the pair is already in $\mathcal{B}$. Otherwise, $\langle m, n \rangle$ is inserted at $H(m, n)$, and we append the information to the adjacency list (i.e., Fig. 7). Edge-edge pairs are processed in the same way but on their own hashing table.

8 EXPERIMENTAL RESULTS

Our implementation platform is a desktop PC with an Intel i9 11900K CPU (8 cores), 128 GB memory, and an Nvidia 3090 GPU. All numerical methods were implemented using C++ on the CPU. We chose Eigen[Guennebaud et al. 2010] and Intel MKL[Wang et al. 2014] as our primary BLAS library and direct linear solvers on the CPU for comparative experiments. Our GPU implementation is matrix-free, and all computations are directly launched on CUDA threads. We use float precision in our GPU solver for large-scale models. Double precision on the GPU is about 30% slower than float with improved robustness for CCD-related computations. Fortunately, as

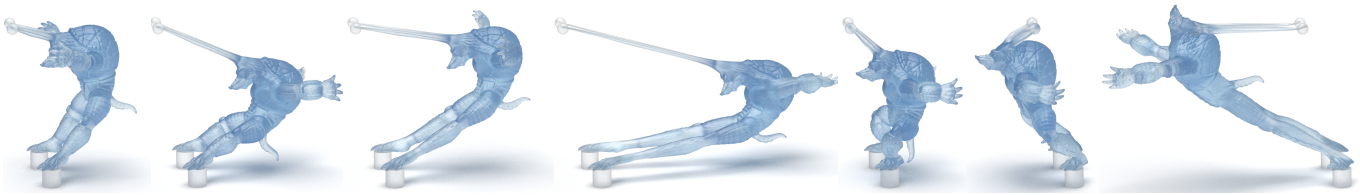

Fig. 9. **Dragging Armadillo.** To objectively examine the convergence of our method, we simulate an elastic Armadillo under large deformations in a collision-free setting using projected Newton (multi-core CPU, MKL), gradient descent (GPU, CUDA), and our method (GPU, CUDA). The model consists of 500K elements and 300K DOFs. Because collisions are ignored, the comparison does not include the warm-start step or local/regular CCD.

our method still carries out a regular line search filtering at the end of each time step, float does not impose any stability issues in our experiments. The CPU benchmark is based on the vanilla open-source IPC implementation [Li et al. 2020, 2021b]. Each Newton solve in CPU IPC is handled with multi-threaded Cholesky factorization from MKL. We note that TBB MKL is $20-50\times$ faster than the LLT solver shipped with Eigen. We use the neo-Hookean material in most experiments, but our method can be applied to any hyperelastic model. Tab 1 gives detailed timing statistics and breakdown of experiments reported in the paper.

8.1 Convergence Benchmark

In the first experiment, we aim to compare our method with some known GPU-based FEM algorithms, such as GPU gradient descent [Wang and Yang 2016]. Since many parts of our method are specially designed for processing barrier functions (e.g., the warm-start step), the comparison naturally favors our method in a collision-rich scenario. To avoid this bias, we compare convergence and computation time in a contact-free setting. As shown in Fig. 9, the Armadillo model consists of 500K elements and 300K DOFs. A sharp dragging force is applied to the back of the Armadillo, with its ears and feet fixed, to trigger large stretching and bending on the body. We compare projected Newton[Teran et al. 2005], Jacobi-preconditioned gradient descent [Wang and Yang 2016], and our method without processing self-collisions. The convergence criterion is set to 10^{-4} of the residual L2 norm for all solvers.

Our observation is consistent with the previous analysis. When time step is small ($\Delta t = 1/100$), and the material is soft (Young's modulus is 5 MPa), the first-order method i.e., Jacobi-preconditioned gradient descent gives the best performance on the GPU, which only uses 0.88 second for one time step. In this setting, Newton's method and our method need 263 seconds and 3.8 seconds respectively. This advantage however, diminishes when the simulation becomes stiffer under a higher Young's modulus or a bigger time step. The total number of the iterations using gradient descent goes up *disproportionately*. For instance, if we increase the time step size from 1/100 to 1/30 second, gradient descent needs 9,628 iterations for one step. Further stiffening the Young's modulus to 30 MPa makes the iteration count jump to over 20K. On the other hand, the projected Newton is quite robust against the variation of Young's modulus and time step sizes. The total number of Newton's iteration increases from 19 to 41 when Δt goes up from 1/100 to 1/30. It takes 581.3 seconds to complete a 1/100 step on average using Newton's method. Setting the Young's modulus to 30 MPa, Newton

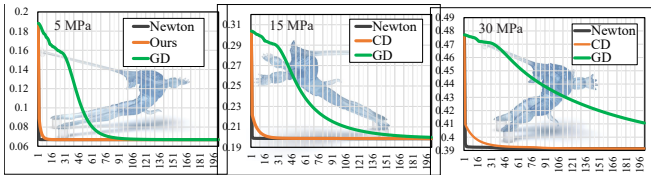

Fig. 10. **Convergence plots.** We plot the convergence curves of Newton's method, gradient descent, and our method for the dragging Armadillo (Fig. 9) under different material stiffness. The curves visualize the variation of total energy at a chosen representative frame of the simulation.

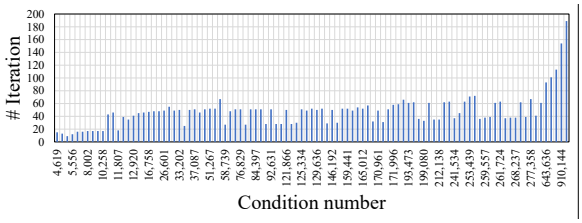

Fig. 11. **Iteration count vs condition number.** We plot the total number of stencil descent iterations when the Hessian matrix has different condition numbers under large deformations.

iteration count reaches 67, and one-time-step computation takes 945.1 seconds.

Compared with gradient descent, our method is locally second-order exhibiting stronger convergence. When Δt increases to 1/30 second, our method surpasses the gradient descent with faster per-step computation (15.3 seconds vs 19.7 seconds). The iteration count of our method does increase for stiffer instances, but the growth rate is milder compared with the first-order approach – from 24 to 73 when Δt is changed from 1/100 to 1/30. Such iteration counts keep CCD manageable when barriers need to be taken care of. With the existence of collisions, our method outperforms gradient descent or other first-order methods by a much bigger margin (i.e., $3\times - 10\times$).

We also plot the convergence curves of three solvers in Fig. 10, where we visualize the variation of the total energy (Eq. (1)) of one frame that needs the most iterations when the Young's modulus is set as 5 MPa, 15 MPa, and 30 MPa. Clearly, the first-order method like gradient descent is sensitive to the stiffness of the simulation. On the other hand, our method uses local curvature information at each stencil making it more resilient to stiffer problems. To further validate this, Fig. 11 shows how iteration counts vary when the

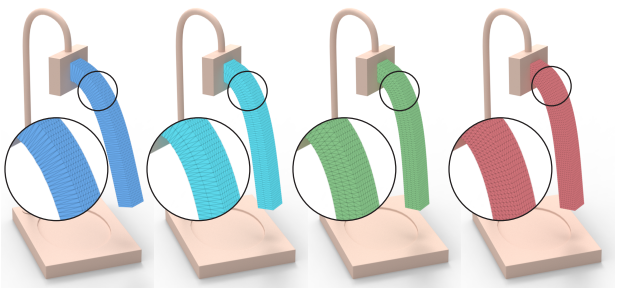

Fig. 12. **The beam model with different tessellations.** The rectangular beam model with different tessellations is bent under gravity. Our solver delivers consistent results with different mesh qualities.

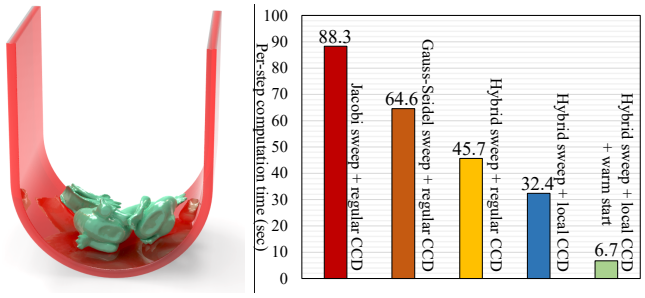

Fig. 13. **Dragon in U.** We conduct an ablation study. The dragon model has one million elements and falls onto a U-shape collider. CPU-based IPC uses 390 sec to simulate one time step ($\Delta t = 1/100$ sec). A Jacobi-like sweep using stencil descent will need 88.3 sec. Switching to Vivace [Fratarcangeli et al. 2016] lows the time to 64.6 sec. Our hybrid sweep strategy further reduces the computation time to 45.7 sec. With the help from local CCD and warm start, we manage to improve the performance to 6.7 sec, which is 13× faster than the naïve implementation.

system matrix has different condition numbers. We evenly sample 100 frames during the simulation of Fig. 9, and corelate the variation of the iteration counts and the corresponding matrix condition number.

Our method produces consistent results under different tessellations. As shown in Fig. 12, we simulate a rectangular beam model discretized with different meshes. From right to left, the ratios between the longest and the shortest edges are 0.7, 0.45, 0.2, and 0.1 respectively. The final equilibrium shapes of the beam however are very similar to each other.

8.2 Ablation Study

Next, we show an ablation study to explain how each module along our pipeline contributes to the overall performance improvement. The study is based on a representative scenario with a dragon hitting a U-shape collider (Fig. 13). The dragon has one million elements and 100K faces on the surface. If we use a Jacobi-style sweep, which solves all stencils in parallel and averages the shared DOFs afterwards, 249 iterations are needed for one step. We also implement Vivace [Fratarcangeli et al. 2016], the state-of-the-art GPU-based Gauss-Seidel coloring, to update the global Δx. Vivace converges faster than the Jacobi sweep using 168 iterations. Nevertheless, our hybrid sweep strategy only needs 97 iterations. In this test, we pre-compute three complementary color schemes, and there are 15 colors in each scheme. We call Jacobi or Gauss-Seidel sweep here to only suggest different strategies to integrate stencil-wise displacements to the global displacement. They should not be confused with linear Jacob or Gauss-Seidel solvers [Fratarcangeli et al. 2016; Wang 2015], which would otherwise need several thousands of iterations for one time step.

The total calculation of our hybrid sweep is heavier than a Gauss-Seidel iteration because we relax stencils in other colors too. Fortunately, those extra computations mostly run at unoccupied CUDA threads. Therefore, we only observe a $3\% - 5\%$ slowdown compared with a Gauss-Seidel sweep.

The adoption of local CCD saves the overall CCD time by 25%. More importantly, local CCD eases the TOI locking induced by small global TOI (i.e., see Fig. 3), which gives 20% overall speedup. Finally, the warm-start step further pushes the computational time to a single digit (6.7 sec). In this example, the warm start only needs three iterations, which takes 88 ms on the GPU. This lightweight process offers a 70% performance jump. As a baseline, the original CPU IPC takes 390 sec to complete one time step for this test, and our method is 58× faster.

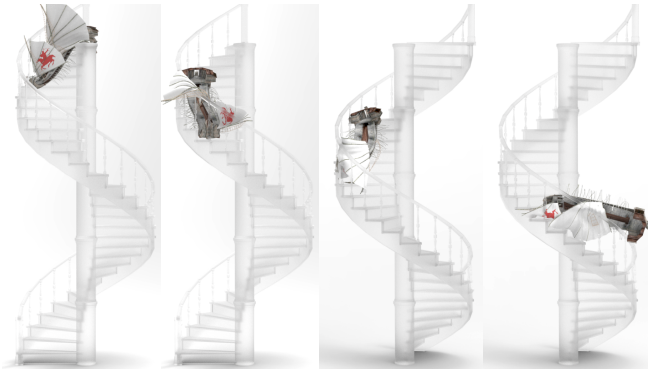

Fig. 14. **A barbarian ship.** A soft barbarian ship with 487K elements falls on the spiral stair. The ship has complex and concave geometry. Its interaction with the stair produces interesting animations. Being an interior-point-based method, our simulation guarantees all triangles on the surface are free of intersection and is 91× faster than the vanilla IPC.

8.3 Efficient Simulation for Complex Collisions

Our method is based on the interior-point method and inherits all the merits of IPC including the algorithmic guarantee of being interpenetration-free. Two such examples are reported here. In Fig. 14, a soft barbarian ship slides down along a spiral stair. The complex geometry of the ship constantly collides and interacts with the handrails of the stair (with 58K triangles on the surface). It eventually gets stuck on the stair by frictional contacts. The barbarian ship has 487K elements, and our method takes 12.5 sec to simulate one step while Newton-based IPC needs about 20 mins on average. This is an over 90× speedup.

Another challenging example is shown in Fig. 15, where a puffer ball falls into an elastic chain net. The net is made of 616 rubber

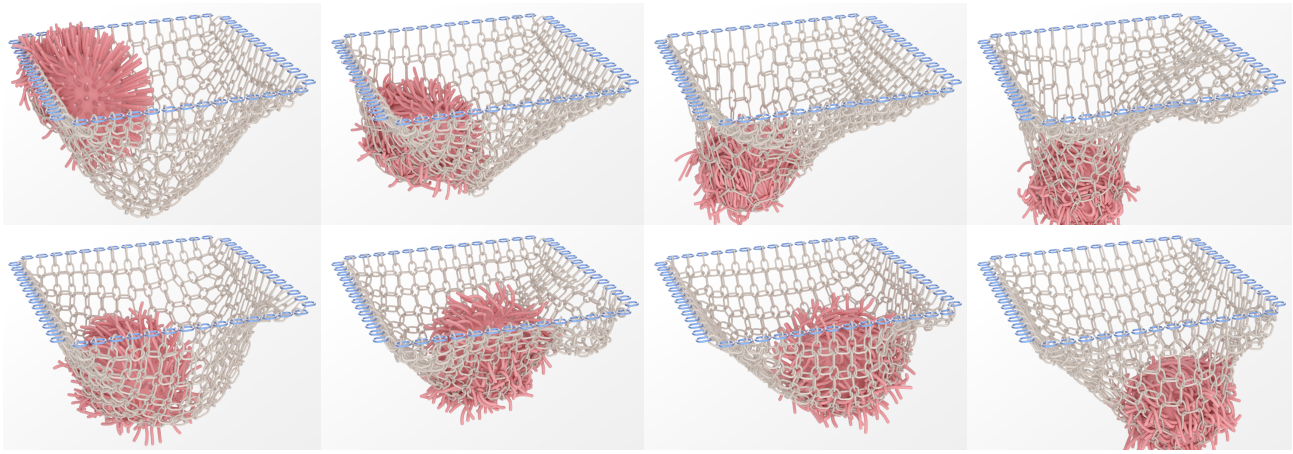

Fig. 15. **A puffer ball meets an elastic chain net.** Contacts and collisions between a soft puffer ball and a dense elastic chain net are difficult to be processed robustly. Many close-to-zero TOIs are generated during the simulation, which cause TOI locking and make CPU IPC cumbersome. Our method delivers both quality and efficiency for this hard simulation problem. In this experiment, the puffer ball has 610K elements, and the net has 344K elements on 616 elastic rings. Our method uses 117 sec on average to simulate one step, while the original IPC needs over 2 hours.

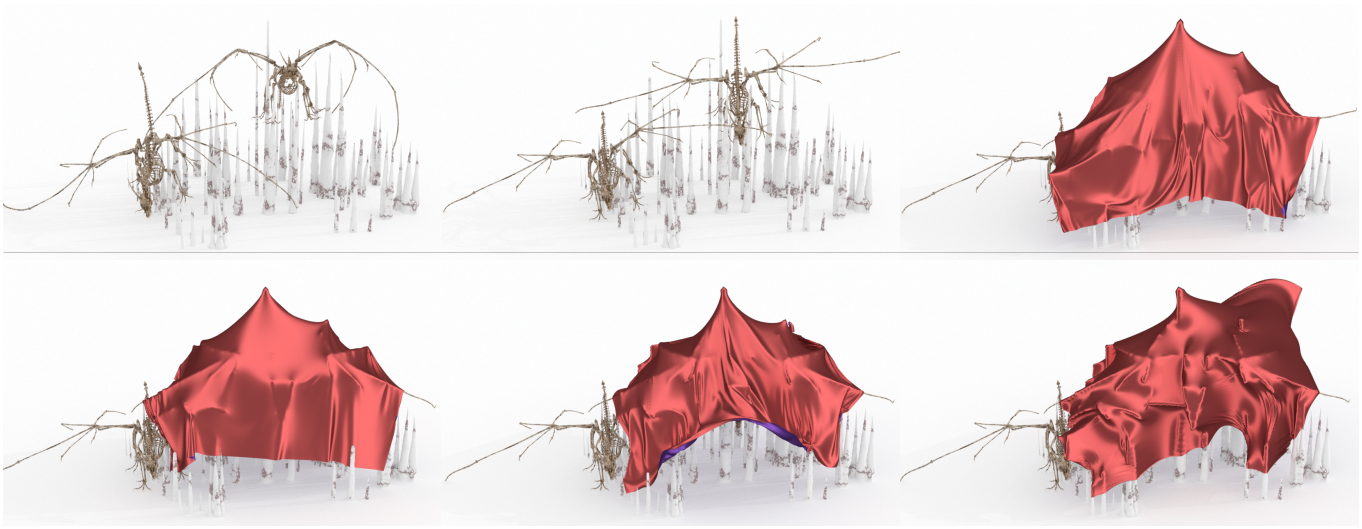

Fig. 16. **Codimensional stencil descent.** Our algorithm is versatile, and can robustly handle codimensional simulations. To validate the capability, we simulate high-resolution interactions between a piece of tablecloth (155K triangles) and two bone dragons (145K elements per dragon) on the spiky surface. The cloth is modeled as a neo-Hookean shell. Under a strong wind field, it gets in contact with the skeleton of the dragon tightly, yielding detailed wrinkles. In this experiment, our method runs 72× faster than codimensional IPC [Li et al. 2021b].

rings, which are mutually coupled via contacts. Each ring has 559 elements and 220 triangles on the surface. Such a network of elastic bodies further interacts with the puffer ball, which has 610K elements and 162K surface triangles. During the simulation, the soft strings on the puffer ball densely entangle the rings on the net. Such interweaving contacts frequently lead to TOI locking and make global displacement update less effective. On the other hand, our method is both robust and efficient. In this example, our method is 58× faster than IPC.

8.4 Versatility for Heterogeneous Models

The good convergence of our method makes it an ideal choice for hybrid models with heterogeneous materials. In Fig. 1, we increase the stiffness of the ship body and add another barbarian ship into the test. If one chooses to use gradient descent [Wang and Yang 2016], the average iteration number will go up by at least one order (well above ten thousand iterations). On the other hand, our method only needs about 100 iterations on average. As we can see from the figure, the animation is of high-quality with an acceleration of two orders (129×) compared with CPU IPC. For this test, our

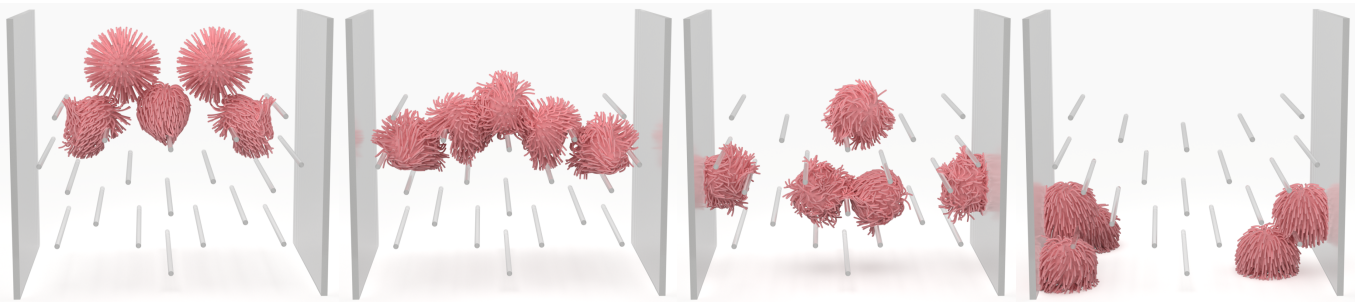

Fig. 17. **Scalability test.** As a stress test, we simulate a large-scale scenario with five falling puffer balls. The total number of elements exceeds three million, and there are over two million simulation DOFs. The simulation of this scale becomes practically impossible using direct solvers even with 128G CPU memory. Therefore, we have to use AMGCL for each Newton solve. On the other hand, our method can deal with this stress test without any problems. On average, our method uses less than five minutes to complete one time step while AMGCL could needs days.

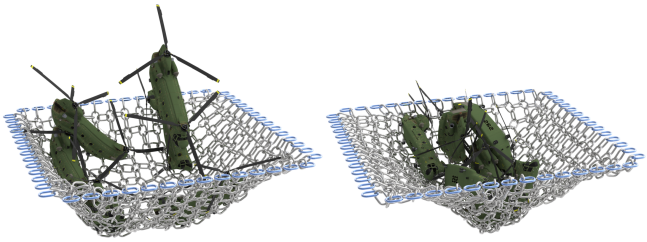

Fig. 18. **Hybrid simulation.** Our method can well handle simulations with both rigid and deformable objects. In this experiment, the chain net is nearly rigid, and each ring is modeled as a stiff affine body. The material on the helicopter is also heterogenous with (20×) stiffer rotor blades and soft cabins. Our method brings a speedup of over 122× faster than CPU based solvers [Lan et al. 2022a].

method is even faster than medial IPC [Lan et al. 2021] by about 30%, which is a reduced simulation algorithm. We find that the conversion between subspace and fullspace coordinates is a major bottleneck for reduced models. However, model reduction simplifies the collision processing and much fewer iterations are needed.

In the experiment of Fig. 18, we make the chain net nearly rigid using affine body dynamics (ABD). The helicopter model also mixes stiff (rotor blades) and soft (the body of the helicopter) components. The blades are 50× stiffer. Each helicopter consists of 280K elements, and there are six helicopters in the test. Our method simulates such hybrid scenario of both rigid and deformable objects efficiently without any interpenetration. In this example, our method is 122× faster than ABD [Lan et al. 2022a] and 15× faster than GPU gradient descent.

Our method can also robustly handle codimensional geometry like thin shell and cloth. As demonstrated in Fig. 16, two bone dragons drop on the spiky terrain. Each bone dragon has 145K elements. After that, we cover the scene with a wind piece of tablecloth with 156K triangles. A strong wild field is then applied pushing cloth tightly in contact with the bone dragon. In this example, we use the full neo-Hookean membrane model for the cloth. Because the simulation involves both tetrahedral and triangular elements, we expand each triangle on the cloth with a virtual vertex so that the local system is still 12-dimension. Doing so balances the computation at threads. In this example, our method is 72× faster than codimensional IPC [Li et al. 2021b].

8.5 Scalability Test

Our method can be conveniently implemented on GPU (or any parallel platforms) in a matrix-free manner. This feature makes it quite scalable. Hereby, we report a stress test of five falling puffer balls (Fig. 17). The total number of elements in this example is over 3.1M. The direct Cholesky factorization becomes extremely slow for this test, which needs dozens of hours to simulate one time step. Therefore, we have to resort to the multigrid method (e.g., AMGCL [Demidov 2019]) for the global Newton solve, which still consumes over 30G CPU memory during the simulation. On the other hand, our solver can right fit into 24G GPU memory of 3090. Under the default AMGCL setting, our method is over 500× faster. This is just an estimation as we are never able to finish this experiment on the CPU using AMGCL. When puffer balls collide with each other, AMGCL will need several *days* to finish the computation of one time step, while our method only needs minutes.

8.6 CPU-GPU Comparison

The speedup of our method does not come from material simplification or early termination of the optimization. In order to justify the accuracy of our method, we carefully compare the result of our method with CPU IPC frame-by-frame and side-by-side. In the example shown in Fig. 19, a long strand of noodle drops into a glass bowl. The noodle has over 342K elements, and it entangles with itself during the falling. As we use a local $\hat{d}$ at each colliding pair (i.e., Eq. (11)), the contact behavior our method is slightly different from the original IPC. The results using both methods are of high quality and free of any interpenetration. Another similar experiment is reported in Fig. 20. Despite small differences of the final contact between the puffer ball and the bottom of the bowl, the resulting animations using our method and CPU IPC are nearly identical.

Lastly, we show a quantitative comparison between our method and CPU IPC to compress a Voronoi cube. The cube embeds an irregular grid. When being compressed, a lot of self-collisions occur. We slowly push the top of the cube downward and compare our

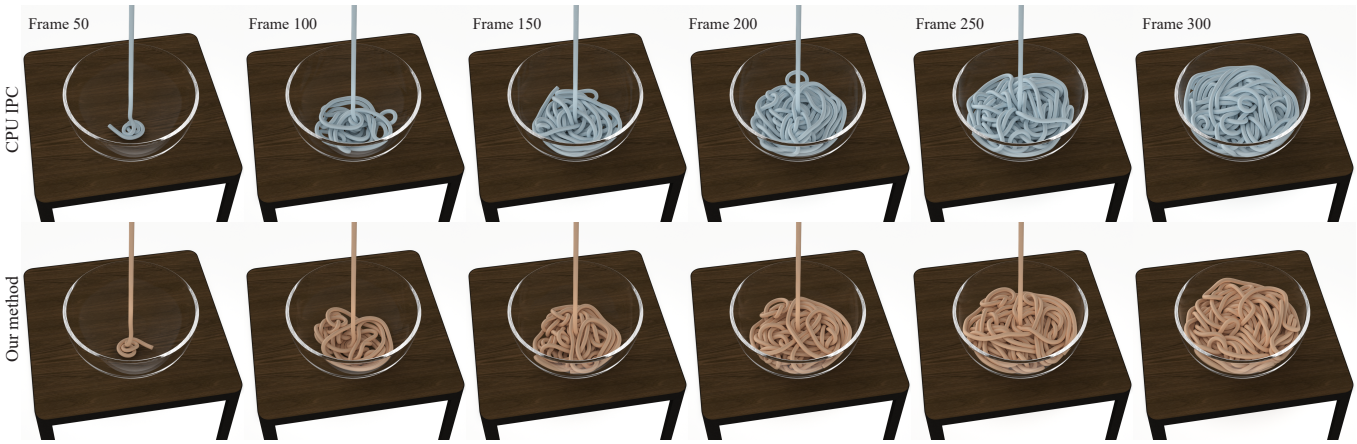

Fig. 19. **A long noodle.** A long strand of noodle drops into a transparent bowl. The noodle has 342K elements. Due to the modification of $\hat{d}$, our simulation (bottom) is slightly different from the original IPC simulation (top). Both animations are plausible and free of any interpenetration. Our method is 25× faster.

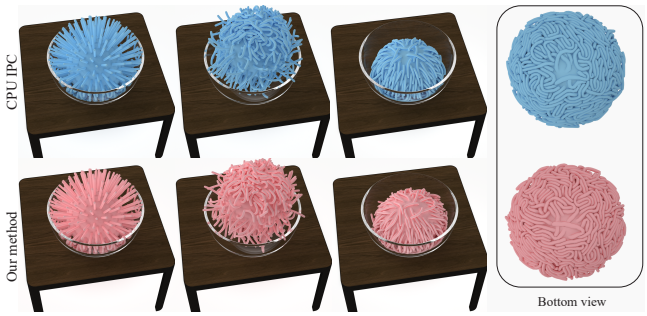

Fig. 20. **A puffer ball in the bowl.** Another side-by-side comparison between CPU IPC and our method. The resulting animations of two falling puffer balls (610K elements) are nearly the same using both methods. From the bottom view, we can see the final contact patterns, when the puffer ball comes to a stop, are slightly different.

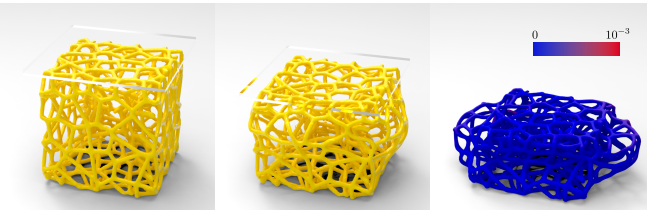

Fig. 21. **Compression comparison.** We quantitatively compare the simulation accuracy using our method with Newton's method when compressing a Voronoi cube. Many self-collisions are generated during the compression, and they are accurately captured by barrier functions. The compression ratio is 60%, and the relative error is visualized using the colormap.

simulation with the results obtained using the Newton's method frame by frame. Per-vertex displacement difference is visualized when the maximum compression (60%) is reached (see Fig. 21, right). Our result is nearly identical to Newton's solution: The maximum relative error is smaller than 10^{-3} but our method is 35× faster.

9 LIMITATION AND CONCLUSION

In this paper, we show that the interior-point method can be well handled and accelerated on the GPU. In elastic body simulation, CCD must be followed after each displacement update to ensure barrier functions are well-defined. Therefore, the GPU interior point calls for a different strategy for convergence-efficiency trade-off. We observe that local second-order methods are effective in relaxing barrier-in-the-loop simulations. While this computation is more costly than the first-order approaches, its improved convergence outweighs this drawback in collision-rich tasks. Based on this parallelization modality, we systematically customize the simulation pipeline including the hybrid sweep scheme that can better harvest the capacity of the modern GPU, the local CCD mechanism that avoids TOI locking, and the warm start that softens barriers before the optimization kicks in.

Our method also has some limitations that lead to several interesting future research directions. First, we note that the first-order method is still competitive when collisions are not massively active. Combining first-order relaxation with second-order ones at different stages of the simulation seems to be a promising idea. Similar to IPC [Li et al. 2020], our method is a primal interior-point implementation. It is known that primal-dual interior point could be more effective for heterogenous systems [Macklin et al. 2020]. We will investigate GPU-based primal-dual interior-point solutions to further enhance the solver's performance. For highly stiff instances, our method may still need a large number of iterations. Subspace preconditioning should be a good remedy to this issue. Our method is more effective and beneficial for large-scale simulations with hundreds of thousands or millions of DOFs. The advantage of our method becomes less obvious for smaller problems. For instance, our method is only about twice faster than CPU IPC for simulations of 10K DOFs. We also note that local relaxation is mathematically equivalent to (nonlinear) convolution, which suggests its close connection to learning-based methods. The potential of using the emerging neural processing unit (NPU) [Yin et al. 2017] for simulation is a worthy future topic.

ACKNOWLEDGMENTS

We thank reviewers for their detailed and constructive comments. Yin Yang is partially supported by NSf under grant numbers of 2301040, 2008915, 2244651, 2008564. Chenfanfu Jiang is supported in part by NSF CAREER 2153851, CCF 2153863, ECCS-2023780.

REFERENCES

Mihai Anitescu and Florian A Potra. 1997. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics* 14, 3 (1997), 231–247.

David Baraff. 1989. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. 223–232.

David Baraff and Andrew Witkin. 1992. Dynamic simulation of non-penetrating flexible bodies. *ACM SIGGRAPH Computer Graphics* 26, 2 (1992), 303–308.

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

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

Jernej Barbič and Doug L James. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. In *ACM Trans. Graph. (TOG)*, Vol. 24. ACM, 982–990.

Jan Bender, Matthias Müller, and Miles Macklin. 2017. A survey on position based dynamics, 2017. *Proceedings of the European Association for Computer Graphics: Tutorials* (2017), 1–31.

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

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

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 594–603.

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

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

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

Kwang-Jin Choi and Hyeong-Seok Ko. 2005a. Research problems in clothing simulation. *Computer-aided design* 37, 6 (2005), 585–592.

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

Jinhyun Choo, Yidong Zhao, Yupeng Jiang, Minchen Li, Chenfanfu Jiang, and Kenichi Soga. 2021. A barrier method for frictional contact on embedded interfaces. arXiv:2107.05814 [math.NA]

Michael B Cline and Dinesh K Pai. 2003. Post-stabilization for rigid body simulation with contact and constraints. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, Vol. 3. IEEE, 3744–3751.

Peter A Cundall and Otto DL Strack. 1979. A discrete numerical model for granular assemblies. *geotechnique* 29, 1 (1979), 47–65.

Ian J. Davis. 1992. A fast radix sort. *The computer journal* 35, 6 (1992), 636–642.

Denis Demidov. 2019. AMGCL: An efficient, flexible, and extensible algebraic multigrid implementation. *Lobachevskii Journal of Mathematics* 40, 5 (2019), 535–546.

Evan Drumwright. 2007. A fast and stable penalty method for rigid body simulation. *IEEE transactions on visualization and computer graphics* 14, 1 (2007), 231–240.

Kenny Erleben. 2007. Velocity-based shock propagation for multibody dynamics animation. *ACM Transactions on Graphics (TOG)* 26, 2 (2007), 12–es.

Yu Fang, Minchen Li, Chenfanfu Jiang, and Danny M Kaufman. 2021. Guaranteed globally injective 3D deformation processing. *ACM Trans. Graph.(TOG)* 40, 4 (2021).

Charbel Farhat, Michael Lesoinne, and Kendall Pierson. 2000. A scalable dual-primal domain decomposition method. *Numerical linear algebra with applications* 7, 7-8 (2000), 687–714.

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

Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M Kaufman, and Daniele Panozzo. 2021. Intersection-free rigid body dynamics. *ACM Transactions on Graphics* 40, 4 (2021), 183.

Susan Fisher and Ming C Lin. 2001. Fast penetration depth estimation for elastic bodies using deformed distance fields. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, Vol. 1. IEEE, 330–336.

Marco Fratarcangeli and Fabio Pellacini. 2015. Scalable partitioning for parallel position based dynamics. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 405–413.

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

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

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

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

Gaël Guennebaud, Benoit Jacob, et al. 2010. Eigen. *URl: http://eigen. tuxfamily. org* (2010).

Shoichi Hasegawa, Nobuaki Fujii, Katsuhito Akahane, Yasuharu Koike, and Makoto Sato. 2004. Real-time rigid body simulation for haptic interactions based on contact volume of polygonal objects. *Transactions of the Society of Instrument and Control Engineers* 40, 2 (2004), 122–131.

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

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

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

Tommy R Jensen and Bjarne Toft. 2011. *Graph coloring problems*. John Wiley & Sons.

Couro Kane, Jerrold E Marsden, Michael Ortiz, and Matthew West. 2000. Variational integrators and the Newmark algorithm for conservative and dissipative mechanical systems. *International Journal for numerical methods in engineering* 49, 10 (2000), 1295–1325.

Danny M Kaufman, Timothy Edmunds, and Dinesh K Pai. 2005. Fast frictional dynamics for rigid bodies. In *ACM SIGGRAPH 2005 Papers*. 946–956.

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

Theodore Kim and Doug L James. 2012. Physics-based character skinning using multidomain subspace deformations. *IEEE transactions on visualization and computer graphics* 18, 8 (2012), 1228–1240.

Laurent Labous, Anthony D Rosato, and Rajesh N Dave. 1997. Measurements of collisional properties of spheres using high-speed video analysis. *Physical review E* 56, 5 (1997), 5717.

Lei Lan, Danny M Kaufman, Minchen Li, Chenfanfu Jiang, and Yin Yang. 2022a. Affine body dynamics: Fast, stable & intersection-free simulation of stiff materials. *arXiv preprint arXiv:2201.10022* (2022).

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

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

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

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. 2020. Incremental potential contact: Intersection- and inversion-free, large-deformation dynamics. *ACM transactions on graphics* 39, 4 (2020).

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

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021b. Codimensional Incremental Potential Contact. *ACM Trans. Graph. (SIGGRAPH)* 40, 4, Article 170 (2021).

Xuan Li, Yu Fang, Minchen Li, and Chenfanfu Jiang. 2021a. BFEMP: Interpenetration-free MPM–FEM coupling with barrier contact. *Computer Methods in Applied Mechanics and Engineering* (2021), 114350.

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

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

Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim. 2020. Primal/dual descent methods for dynamics. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 89–100.

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

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*. 49–54.

Miles Macklin, Kier Storey, Michelle Lu, Pierre Terdiman, Nuttapong Chentanez, Stefan Jeschke, and Matthias Müller. 2019. Small Steps in Physics Simulation. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, Christopher Batty and Jin Huang (Eds.). ACM. https://doi.org/10.1145/3309486.3340247

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

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-based elastic materials. In *ACM SIGGRAPH 2011 papers*. 1–8.

Sanjay Mehrotra. 1992. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization* 2, 4 (1992), 575–601.

Ullrich Meier, Oscar López, Carlos Monserrat, Mari C Juan, and M Alcaniz. 2005. Real-time deformable models for surgery simulation: a survey. *Computer methods and programs in biomedicine* 77, 3 (2005), 183–197.

Brian Mirtich and John Canny. 1995. Impulse-based simulation of rigid bodies. In *Proceedings of the 1995 symposium on Interactive 3D graphics*. 181–ff.

Matthew Moore and Jane Wilhelms. 1988. Collision detection and response for computer animation. In *ACM Siggraph Computer Graphics*, Vol. 22. ACM, 289–298.

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

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.

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

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

Alexander Naitsat, Yufeng Zhu, and Yehoshua Y. Zeevi. 2020. Adaptive Block Coordinate Descent for Distortion Optimization. *Computer Graphics Forum* 39, 6 (2020), 360–376.

Rahul Narain, Matthew Overby, and George E Brown. 2016. ADMM ⊇ projective dynamics: fast simulation of general constitutive models.. In *Symposium on Computer Animation*, Vol. 1. 2016.

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

V Popescu, Grigore Burdea, and Mourad Bouzit. 1999. Virtual reality simulation modeling for a haptic glove. In *Proceedings Computer Animation 1999*. IEEE, 195–200.

Siyuan Shen, Yang Yin, Tianjia Shao, He Wang, Chenfanfu Jiang, Lei Lan, and Kun Zhou. 2021. High-order Differentiable Autoencoder for Nonlinear Model Reduction. *arXiv preprint arXiv:2102.11026* (2021).

Jonathan Richard Shewchuk et al. 1994. An introduction to the conjugate gradient method without the agonizing pain.

Eftychios Sifakis and Jernej Barbic. 2012. FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*. ACM, 20.

Horst D Simon. 1992. *Parallel computational fluid dynamics-implementations and results*. Technical Report. National Aeronautics and Space Administration, Moffett Field, CA (United ....

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

Min Tang, Dinesh Manocha, Miguel A Otaduy, and Ruofeng Tong. 2012. Continuous penalty forces. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–9.

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*. 181–190.

Demetri Terzopoulos and Kurt Fleischer. 1988. Deformable models. *The visual computer* 4, 6 (1988), 306–331.

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically deformable models, In Proceedings of the 14th annual conference on Computer graphics and interactive techniques. *ACM Siggraph Computer Graphics* 21, 4, 205–214.

Demetri Terzopoulos, Andrew Witkin, and Michael Kass. 1988. Constraints on deformable models: Recovering 3D shape and nonrigid motion. *Artificial intelligence* 36, 1 (1988), 91–123.

Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger, Gabriel Zachmann, Laks Raghupathi, Arnulph Fuhrmann, M-P Cani, François Faure, Nadia Magnenat-Thalmann, Wolfgang Strasser, et al. 2005. Collision detection for deformable objects. In *Computer graphics forum*, Vol. 24. Wiley Online Library, 61–81.

Quoc-Minh Ton-That, Paul G Kry, and Sheldon Andrews. 2022. Parallel block Neo-Hookean XPBD using graph clustering. *Computers & Graphics* (2022).

Quoc-Minh Ton-That, Paul G Kry, and Sheldon Andrews. 2023. Parallel block Neo-Hookean XPBD using graph clustering. *Computers & Graphics* 110 (2023), 1–10.

Takuya Umedachi, Vishesh Vikas, and Barry A Trimmer. 2013. Highly deformable 3-D printed soft robot generating inching and crawling locomotions with variable friction legs. In *2013 IEEE/RSJ international conference on Intelligent Robots and Systems*. IEEE, 4590–4595.

Juraj Vanek, Jorge A Garcia Galicia, and Bedrich Benes. 2014. Clever support: Efficient support structure generation for digital fabrication. In *Computer graphics forum*, Vol. 33. Wiley Online Library, 117–125.

Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. 2014. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi™*. Springer, 167–188.

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

Huamin Wang. 2018. Rule-free sewing pattern adjustment with precision and efficiency. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–13.

Huamin Wang and Yin Yang. 2016. Descent methods for elastic body simulation on the GPU. *ACM Trans. Graph. (TOG)* 35, 6 (2016), 212.

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.

Rachel Weinstein, Joseph Teran, and Ronald Fedkiw. 2006. Dynamic simulation of articulated rigid bodies with contact and collision. *IEEE Transactions on Visualization and Computer Graphics* 12, 3 (2006), 365–374.

Dominic JA Welsh and Martin B Powell. 1967. An upper bound for the chromatic number of a graph and its application to timetabling problems. *Comput. J.* 10, 1 (1967), 85–86.

Louis J Wicker and William C Skamarock. 2002. Time-splitting methods for elastic models using forward time schemes. *Monthly weather review* 130, 8 (2002), 2088–2097.

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

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

Longhua Wu, Botao Wu, Yin Yang, and Huamin Wang. 2020. A Safe and Fast Repulsion Method for GPU-based Cloth Self Collisions. *ACM Transactions on Graphics (TOG)* 40, 1 (2020), 1–18.

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

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

Hongyi Xu, Yili Zhao, and Jernej Barbič. 2014. Implicit multibody penalty-baseddistributed contact. *IEEE transactions on visualization and computer graphics* 20, 9 (2014), 1266–1279.

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

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

Shouyi Yin, Peng Ouyang, Shibin Tang, Fengbin Tu, Xiudong Li, Leibo Liu, and Shaojun Wei. 2017. A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications. In *2017 Symposium on VLSI Circuits*. IEEE, C26–C27.

Cem Yuksel. 2022. A Fast & Robust Solution for Cubic & Higher-Order Polynomials. In *ACM SIGGRAPH 2022 Talks*. 1–2.

Xin-Yuan Zhao, Defeng Sun, and Kim-Chuan Toh. 2010. A Newton-CG augmented Lagrangian method for semidefinite programming. *SIAM Journal on Optimization* 20, 4 (2010), 1737–1765.

Jun Zheng, Suhail S Saquib, Ken Sauer, and Charles A Bouman. 2000. Parallelizable Bayesian tomography algorithms with rapid, guaranteed convergence. *IEEE Transactions on Image Processing* 9, 10 (2000), 1745–1759.

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