Progressive Shell Quasistatics for Unstructured Meshes

JIAYI ERIS ZHANG, Adobe, USA and Stanford University, USA
JÉRÉMIE DUMAS, Adobe, USA
YUN (RAYMOND) FEI, Adobe, USA
ALEC JACOBSON, Adobe, Canada and University of Toronto, Canada
DOUG L. JAMES, Stanford University, USA
DANNY M. KAUFMAN, Adobe, USA

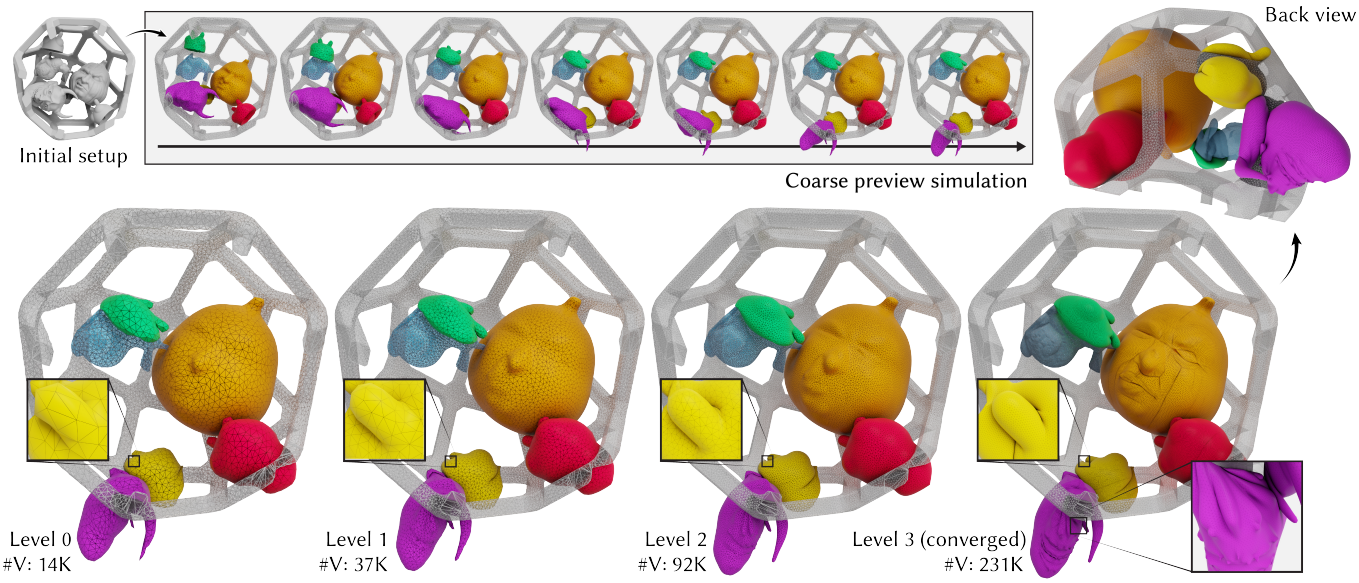


Fig. 1. **Dreaming of Progressive Shell Quasistatics (PSQ):** Progressive simulation of sleepy shell characters resting inside a rigid fullerene shape. Both the rigid colliders (bunny and cage) and the balloon-like characters are modeled using unstructured meshes, which are coarsened, posed by an artist, and then progressively and safely refined (Left to Right) during PSQ simulation. (Left) Our fast, coarse-mesh PSQ approximation is an excellent predictor across simulation scales, and faithfully represents the (Middle) intermediate-resolution solution and the (Right, Far-Right) converged fine-scale solution complete with deformed character details and wrinkles. Despite these benefits, the coarse PSQ proxy is over two orders of magnitude faster to simulate than its detailed counterpart. Sweet dreams.

Thin shell structures exhibit complex behaviors critical for modeling and design across wide-ranging applications. Capturing their mechanical response requires finely detailed, high-resolution meshes. Corresponding simulations for predicting equilibria with these meshes are expensive, whereas coarse-mesh simulations can be fast but generate unacceptable artifacts and inaccuracies. The recently proposed progressive simulation framework [Zhang et al. 2022] offers a promising avenue to address these limitations with consistent and progressively improving simulation over a hierarchy of increasingly higher-resolution models. Unfortunately, it is currently severely limited in application to meshes and shapes generated via Loop subdivision.

We propose Progressive Shells Quasistatics to extend progressive simulation to the high-fidelity modeling and design of all input shell (and plate) geometries with unstructured (as well as structured) triangle meshes. To do so, we construct a fine-to-coarse hierarchy with a novel nonlinear prolongation operator custom-suited for curved-surface simulation that is rest-shape preserving, supports complex curved boundaries, and enables the reconstruction of detailed geometries from coarse-level meshes. Then, to enable convergent, high-quality solutions with robust contact handling, we propose a new, safe, and efficient shape-preserving upsampling method that ensures non-intersection and strain limits during refinement. With these core contributions, Progressive Shell Quasistatics enables, for the first time, wide generality for progressive simulation, including support for arbitrary curved-shell geometries, progressive collision objects, curved boundaries, and unstructured triangle meshes – all while ensuring that preview and final solutions remain free of intersections. We demonstrate these features

Authors' addresses: Jiayi Eris Zhang, Adobe, USA, Stanford University, USA, eriszhan@stanford.edu; Jérémie Dumas, Adobe, USA, jeremie.dumas@ens-lyon.org; Yun (Raymond) Fei, Adobe, USA, yfei@adobe.com; Alec Jacobson, Adobe, Canada, University of Toronto, Canada, jacobson@cs.toronto.edu; Doug L. James, Stanford University, USA, djames@cs.stanford.edu; Danny M. Kaufman, Adobe, USA, dannykaufman@gmail.com.

0730-0301/2023/12-ART184 $15.00
https://doi.org/10.1145/3618388

across a wide range of stress-tests where progressive simulation captures the wrinkling, folding, twisting, and buckling behaviors of frictionally contacting thin shells with orders-of-magnitude speed-up in examples over direct fine-resolution simulation.

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

Additional Key Words and Phrases: Progressive Simulation, Multiresolution, Model Reduction, Shell Simulation, Contact Mechanics

ACM Reference Format:
Jiayi Eris Zhang, Jérémie Dumas, Yun (Raymond) Fei, Alec Jacobson, Doug L. James, and Danny M. Kaufman. 2023. Progressive Shell Quasistatics for Unstructured Meshes. *ACM Trans. Graph.* 42, 6, Article 184 (December 2023), 17 pages. https://doi.org/10.1145/3618388

1 INTRODUCTION

Thin-shell structures formed from combinations of curved and flat geometries exhibit complex and often surprising nonlinear behaviors critical for modeling and designing materials in diverse applications ranging from engineering and robotics to fashion and entertainment. Capturing the mechanical response of these materials requires well-shaped elements on finely detailed, carefully constructed, high-resolution meshes. Corresponding simulation times for computing shell equilibria with these meshes are then impractical, while fast simulations on coarse proxy meshes generate significant numerical artifacts with unacceptable inaccuracies.

Zhang et al's [2022] recently proposed Progressive Cloth Simulation (PCS) method constructs a promising progressive simulation framework to address these limitations with consistent and progressively improving simulation over a hierarchy of increasingly higher-resolution models. Unfortunately, progressive simulation is currently severely limited in its application by virtue of being restricted to the simulation of just meshes and geometries that can be generated via Loop subdivision. In turn, this coarse-to-fine strategy currently prohibits the progressive simulation of high-resolution models with intricate geometric details, complex boundaries, and/or high-quality unstructured meshes.

We propose Progressive Shell Quasistatics to extend progressive simulation to support the high-fidelity modeling and design of all input shell (as well as plate and cloth) geometries and their corresponding triangle meshes. Progressive Shell Quasistatics provides fast progressive simulation and final high-resolution, converged simulation results that respect user-input triangle-mesh topologies (both unstructured and structured) and their detailed geometries (see Figure 1).

To enable these features, Progressive Shell Quasistatics addresses three core challenges. First, the progressive simulation solver must now be capable of resolving solutions on high-resolution models with intricate boundaries, geometric details, non-zero curvatures, and unstructured triangulations. To do so, we construct a decimation-based, top-down shell-simulation hierarchy via recursive edge-collapse [Garland and Heckbert 1997]. Second, this fine-to-coarse hierarchy requires a custom-suited prolongation operator to map displacements and restrict forces between mesh levels. As we cover in §4 and §5, prior prolongation methods for fine-to-coarse hierarchies are unsuitable for shell simulation and instead generate unacceptable artifacts and prohibit convergence. To address this challenge, we build a new, nonlinear prolongation operator, custom-suited for curved shell geometry simulation and rest-shape preservation. Third, to support convergent solutions and robust contact handling, progressive simulation requires a prolonged, interpenetration-free initializer to bootstrap simulation solves at each new level of refinement. While the nested hierarchy produced by PCS's subdivision operator offers simple and direct contact-safe initialization via in-plane upsampling, there is no corresponding method available for intersection-free upsampling via vertex expansion. For safe initialization, we propose an efficient, parallel shape-preserving upsampling method that ensures non-intersection (and strain-limiting) feasibility for each step of prolongation in our fine-to-coarse hierarchy.

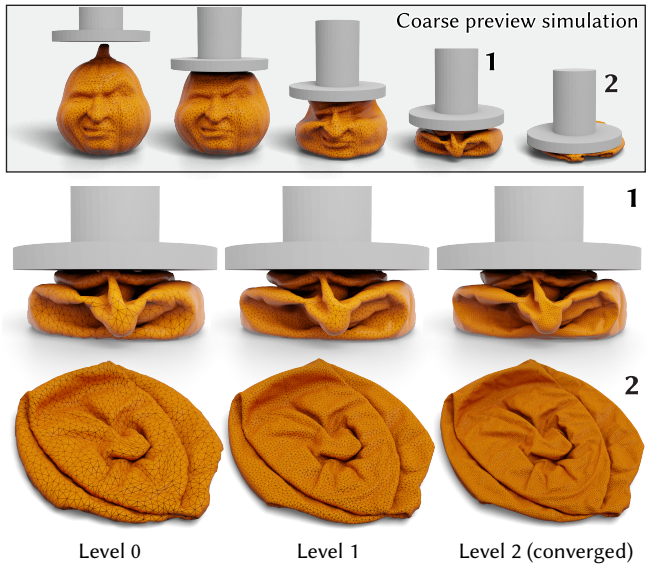


Fig. 2. **Unimpressed Pumpkin Press:** (Top) frames from a coarse-level PSQ preview simulation of a press crushing a thin-wall aluminum pumpkin. (Bottom row, left-to-right) at any step of coarse-level preview (here, the last two frames), PSQ can pause to compute consistent, progressive simulation results for each level of refinement, up to the final, converged, finest-scale solutions.

1.1 Contributions

In summary, Progressive Shell Quasistatics now enables, for the first time, wide generality for progressive simulation, including support of curved shells, arbitrary collision object geometries, curved boundaries (both for thin plate and shell models), and unstructured triangle meshes–all while ensuring that both preview and final solutions remain free of intersections. Our technical contributions include

- Enabling progressive simulation on all input high-resolution triangular shell models via the construction of decimation-based, fine-to-coarse hierarchies;

- A new, nonlinear prolongation operator carefully designed to address curved shell geometry simulation with rest-shape preservation;
- A safe, shape-preserving upsampling method via successive edge expansion that ensures non-intersection (and strain-limiting) feasibility for each refinement level in a decimation-constructed fine-to-coarse hierarchy and
- An efficient parallel algorithm for safe expansion.

In extensive evaluation and comparison, across a wide range of stress-tests and examples with widely ranging meshes, geometries, and material properties, we demonstrate that Progressive Shell Quasistatics captures the intricate wrinkling, folding, twisting, and buckling behaviors of frictionally contacting thin-shell structures with over orders-of-magnitude speed-up over direct fine-resolution simulation.

2 RELATED WORK

2.1 Shell Mechanics

The mechanical simulation of thin shells has received enormous attention in computer graphics and engineering, particularly for cloth modeling [Baraff and Witkin 1998; Bridson et al. 2002; Grinspun et al. 2003; Harmon et al. 2009; Li et al. 2020b; Narain et al. 2012; Terzopoulos et al. 1987; Volino and Thalmann 2000]. Decades of progress has led to a collection of successful shell simulation techniques, such as implicit time-integration methods [Baraff and Witkin 1998; Bridson et al. 2002; Kim 2020; Li et al. 2020b, 2018; Narain et al. 2012; Otaduy et al. 2009; Tang et al. 2016, 2018], collision processing and strain-limiting methods [Bridson et al. 2002; Goldenthal et al. 2007; Harmon et al. 2009, 2008; Li et al. 2018, 2021; Narain et al. 2013, 2012; Vouga et al. 2011], and constitutive modeling [Chen et al. 2018b; Clyde et al. 2017; Guo et al. 2018; Jiang et al. 2017; Miguel et al. 2012; Narain et al. 2012; Weischedel 2012]. The demand for faster simulation has resulted in new solver formulations [Bender et al. 2013; Bouaziz et al. 2014; Daviet 2020; Ly et al. 2020; Zhang et al. 2019], and high-performance parallel implementations [Li et al. 2020b; Schmitt et al. 2013; Selle et al. 2008; Tang et al. 2013, 2016, 2018; Wang 2021]. The trade-off between high-fidelity simulation and interactivity is confounding for interactive design and animation tools [Designer 2022; SideFX 2022]. Progressive cloth simulation (PCS) [Zhang et al. 2022] can enable the best of both worlds, with fast interactive design using faithful coarse proxy models and subsequent refinement at improved speeds to final high-resolution simulation. However, as covered above, progressive simulation in its current form has remained highly limited in application when it comes to modeling structures beyond flat sheets.

The progressive refinement strategy we adopt (and as applied in PCS [Zhang et al. 2022]) is originally inspired by Sensitive Couture's [Umetani et al. 2011] hierarchical and successive improvement algorithm. However, following PCS, we use fine-scale force evaluations, which encourage consistency across increasing resolution models. This is key to avoiding the significant artifacts and inconsistencies that are generated by Sensitive Couture[Zhang et al. 2022], and instead provides faithful coarse-scale previews and consistent refinement across resolutions. Multi-scale hierarchies have also long been applied for multigrid methods to accelerate linear system solves when simulating shell models [Tamstorf et al. 2015; Wang et al. 2018; Xian et al. 2019]. Such multigrid solvers and preconditioners are complementary to Progressive Shell Quasistaics, and could be applied to accelerate the solution of linear systems arising within Newton iterations at each level of the progressive solver.

2.2 Coarse-to-Fine Surface Hierarchies

Subdivision surfaces are a classical multiresolution mesh hierarchy for modeling piecewise smooth surfaces using coarse-scale geometry and edits [Zorin et al. 2000]. Linear subdivision methods naturally provide linear prolongation operators to map quantities between levels. The subdivision exterior calculus [De Goes et al. 2016] extends these operators to differential forms. However, these are inherently limited to meshes defined by coarse input cages, with correspondingly cage-scale details and boundaries. Nonetheless, pioneering work on multiresolution surface editing of detailed meshes with subdivision connectivity was introduced by Zorin et al. [1997], with shape edits achieved using coarse-scale detail modifications. The starting point for our work, Zhang et al. [2022], utilizes subdivided planar meshes, and works with this category of coarse-to-fine hierarchies. Application of that work to *curved* subdivision surfaces disappointingly limits input and simulation domains to smooth shapes defined by coarse cages. This simply constrains the hierarchy applied in the progressive simulation, to the hierarchy used to define the geometry domain. Our essential contributions thus reformulate Zhang et al.'s [2022] progressive simulation framework to work in a fine-to-coarse manner, and generalize its success on cloth and plates with simple boundaries, to arbitrary surface shapes and unstructured triangle meshes, with possibly complex boundaries, by decoupling the input shape representation from the progressive simulation hierarchy.

2.3 Fine-to-Coarse Surface Hierarchies

We build on ideas from Progressive Meshes [Hoppe 1996] and Kobbelt et al. [1999] for building fine-to-coarse unstructured mesh hierarchies using recursive edge-collapse operations [Garland and Heckbert 1997]. Later, multiresolution modeling on *arbitrary* meshes was explored by Kobbelt et al. [1998], who used a decimation- and smoothing-based approach to multiresolution hierarchy construction and detail estimation, with benefits for smoothing and fairing applications. Their details were encoded in local frames not based at a vertex or single face, but on local low-order polynomial interpolants or approximants that depend on more than one triangle. In our approach, we use local per-face frames which are more convenient, e.g., for gradient computation, and we avoid smoothing to produce faithful coarse-scale approximations for simulation and contact. Normal meshes [Guskov et al. 2000] are multiresolution meshes where each level can be written as a normal offset from a coarser version, using just a single float per vertex, which is useful for compression but not for deformable simulation. Displaced subdivision surfaces [Lee et al. 2000] allow deformable simulations at coarser scales to be displaced post-subdivision, e.g., James and Pai [2003], but fundamentally ignore fine-scale physics.

Multi-scale decompositions — such as the displacement volumes of Botsch and Kobbelt [2003] — separate coarse surface changes from

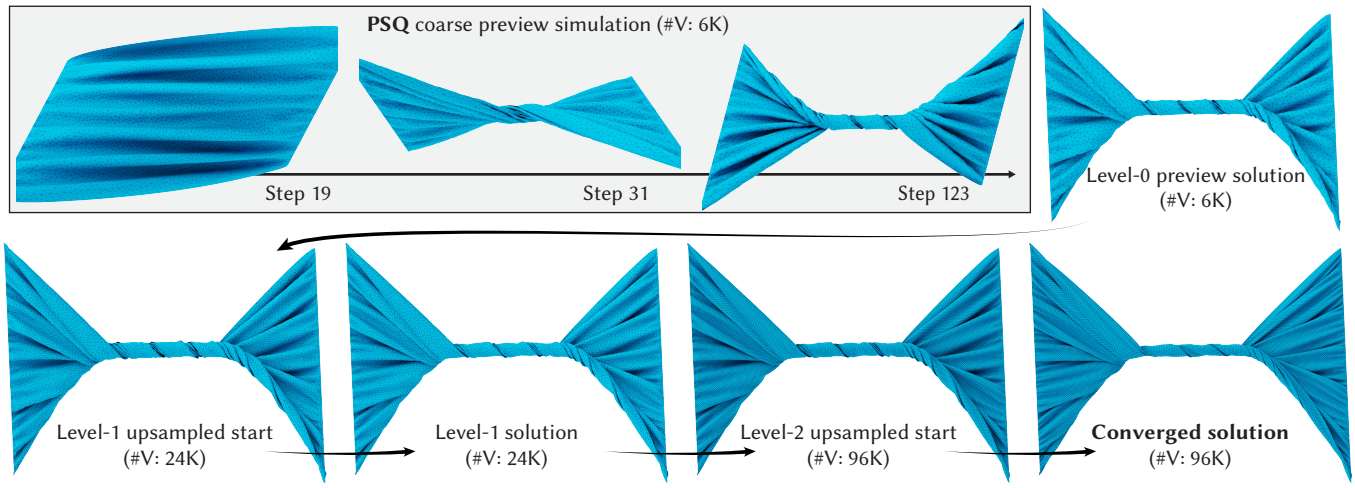


Fig. 3. **Progressive Steps in Shell Quasistatic Simulation:** (Left-to-right) we demonstrate the alternating steps of PSQ's computation of a simulation solution per level, followed by each solution's prolongation to a safely initialized, upsampled geometry to start the next level's simulation solve. (Top) Coarsest-level simulation quickly evolves the solution to an equilibrium predictive of the fine-scale converged solution. (Bottom) Repeated prolongations and solves per level then rapidly converge to an interpenetration- and artifact-free, finest-level solution demonstrating intricate twisting, folding, and self-contact.

high-frequency details during editing. Special care is paid to ensure self-intersection-free editing. The propagation of coarse-level edits to the fine-scale (or equivalently, the re-introduction of fine-scale details onto a deformed coarse surface) requires a mapping between the coarse and fine surfaces. Avoiding problems related to normal offsets [Guskov et al. 2000], Manson and Schaefer [2011] build this mapping via a best-fit rigid transformation computed during a reversal of edge-collapse operations. They use this mapping to transfer as-rigid-as-possible deformations from a decimated model to its fine mesh original. In response to the nonlinearity of this mapping, Liu and Jacobson [2019] use a best-fit affine transformation for reversing each edge-collapse, producing a linear prolongation operator. While these operators can capture some local deformation and recover the rest shape, as we demonstrate in §5, they are not suitable for the progressive simulation of bending and buckling shells.

Our hierarchy construction is inspired by the *intrinsic* prolongation method [Liu et al. 2021], which in turn traces its roots to MAPS [Lee et al. 1998]. Liu et al. [2021] (and very recently Liu et al. [2023]) improve on the simple one-ring averaging of Aksoylu et al. [2005] by tracking a bijective mapping between local patches during edge-collapses. These local maps can be composed into a global mapping between entire hierarchy levels. This construction, however, is only suitable for intrinsic quantities and can not recover rest shape, while shell modeling must be able to capture *extrinsic* bending and rest-shape.

2.4 Coarsening and Homogenization

Our method's preview-mode simulation of a low-resolution discrete deformable model that behaves similarly to a higher-resolution model is related, in goal, to the numerical coarsening of mechanical systems. Some coarsening strategies consider the homogenization of spatially varying elastic materials, and seek to estimate general (e.g., anisotropic) material parameters for coarsened elements such that the coarse-scale elastic response approximates the fine-scale one without direct evaluation on fine-scale degrees of freedom [Chen et al. 2018a; Kharevych et al. 2009; Schumacher et al. 2015]. In computer graphics, other works also seek to construct coarsened models to estimate the behavior of complex embedded geometries [Nesme et al. 2009].

In this work, we do not seek novel constitutive models for coarsened shell elements. Instead, akin to embedding, we rely on prolongation to deform the fine-scale geometry using the coarse proxy. Here, this enables us to estimate the implied response of the fine model. During each level's preview solve our progressive simulation approach can then loosely be viewed [Zhang et al. 2022] as a form of multi-scale dimensional model reduction (DMR) [Grinspun et al. 2002; Krysl et al. 2001] for coarsening shell simulation models while retaining fine-scale awareness of internal elastic energies. Unlike works that seek to avoid fine-scale force evaluation, with pre-computed subspace-only force models [An et al. 2008; Barbič and James 2005] or other evaluation speedups [Chaturantabut and Sorensen 2010], we leverage prolongation to evaluate fine-scale forces and fast multi-scale restriction to project to the coarser levels, and, at the same time, target rapid, improved speeds to reach a final simulation converged on the full degree-of-freedom mesh. As with classical implicit subspace integration methods for DMR [Krysl et al. 2001] we (parallel) evaluate fine-scale forces, and our speedup, per level, is primarily due to the smaller systems of equations to solve during quasistatic time stepping.

In other multi-scale model reduction works, the focus is on spatial adaptivity, and the projection of fine-scale forces is avoided using adaptive quadrature schemes [Grinspun et al. 2002]. Here we focus on quickly computing high-quality coarse previews before gaining further speed-up by progressively and rapidly refining to the finest-level solution.

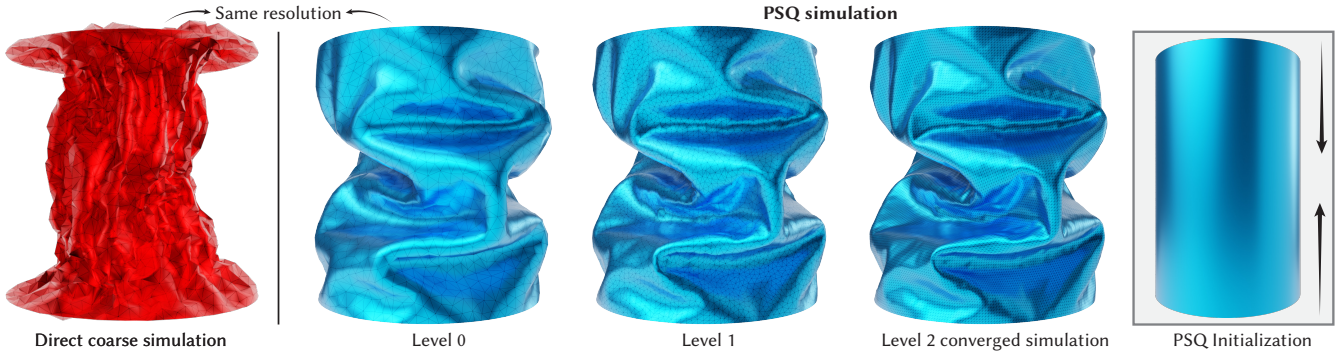


Fig. 4. **Coarse shell models that can!** (Left, red) Direct coarse-level simulation of a crushed, thin-wall aluminum cylinder exhibits severe locking artifacts. In contrast, (right, blue) our coarse-level PSQ simulation, using the same base mesh, generates a predictive solution at a comparable speed, that is consistent with (farther right) its mid-level prediction *and* its finest-level, converged result.

3 BACKGROUND: PROGRESSIVE SIMULATION

Zhang et al. [2022] introduce the progressive simulation framework for the fast preview and efficient solution of high-quality shell quasistatics simulations on triangle-meshed geometries. Progressive simulation generates consistent and improving solution previews via a hierarchy of increasingly higher-resolution meshes, with converged simulation output generated on a final, high-resolution mesh.

This hierarchy is constructed from a set of triangle meshes and a corresponding set of prolongation operators each mapping to the next finer-resolution mesh. Meshes in the hierarchy are indexed in increasing resolution by subscript $l \in [0, L]$, where x_l and $\bar{x}_l \in \mathbb{R}^{3n_l}$ are, respectively, the n_l deformed and rest positions of mesh nodes at level l. Deformed positions of the coarsest mesh are then stored in x_0, while x_L gives the finest-resolution positions of the final, converged, high-quality simulation output. In turn, each level l is associated with a prolongation operator, P^l_{l+1}, mapping nodal positions from the current level to the next level $l+1$. To simplify the discussion, where clear we will designate finest-level resolution quantities without decoration, so that, e.g., $x = x_L$, $\bar{x} = \bar{x}_L$, and $n = n_L$.

We equip each simulation mesh[1] with shell (Ψ), contact barrier (B), friction (D), and, when required, strain-limiting potential energies (S) to compute the stable *equilibria* of frictionally contacting shells subject to imposed boundary conditions and external forces. These are the local (constrained) minimizers of the total potential energy, $E_l(x) = E_l(x, \bar{x}, u)$ constructed from the sum of the above potentials, $E_l = \Psi_l + B_l + D_l + S_l$, where u collects current material and boundary-condition parameters.

Given a starting nonequilibrium configuration, x^t, and current parameters, u, progressive simulation computes stable equilibria by time-stepping its gradient flow with implicit Euler. At the finest level, this amounts to computing a sequence of forward quasistatic positional updates via artificial "time steps" (of size h) from t to $t+1$ via the minimization of an updated *incremental* potential,

$$x^{t+1} = \underset{x}{\operatorname{argmin}} \frac{1}{2h^2}\|x - x^t\|^2_M + E(x, \bar{x}, u^{t+1}), \tag{1}$$

with mass matrix M, until convergence to equilibrium, given by $\|\nabla E(x^*)\| \leq \epsilon$, is satisfied.

To avoid expensive direct simulation on the target fine-resolution mesh, progressive simulation applies its hierarchy in a one-way, nonlinear multiresolution simulation-solver [Zhang et al. 2022] with two solution phases:

Preview, $x^t_l \to x^{t+1}_l$**:** quasistatic advancement of the solution, at level l, from time step t to $t+1$, over possibly varying u to provides consistent solution previews at each level; and

Refinement, $x^{t+1}_l \to x^{t+1}_{l+1}$**:** progressive spatial improvement of the solution from level l to $l+1$, for a fixed set of conditions u^t.

3.1 Coarse-Level Proxy Energies

At each coarsened-level $l < L$, previews (over varying parameters u) and progressively refined solutions are made consistent by solving each time step's quasistatics with a *proxy* for the finest-level potential energy,

$$F_l(x_l) = \underbrace{B_l(x_l) + D_l(x_l) + S_l(x_l)}_{C_l(x_l)} + \Psi_l\big(P^l(x_l)\big). \tag{2}$$

Here shell elastics, Ψ_l, are evaluated at the finest-resolution model, via a direct prolongation, $P^l(x_l)$, from level l up to the finest scale, while coarse, barrier-based potential terms in $C_l(x_l)$ enforce contact and strain-limit feasibility on the current level-l's geometry. Each coarse level is then solved by stepping via

$$x^{t+1}_l = \underset{x_l}{\operatorname{argmin}} \frac{1}{2h^2}\|x_l - x^t_l\|^2_{M_l} + F_l(x_l). \tag{3}$$

3.2 Refinement and Safe Initialization

When our preview of the solution at level $l < L$ is finalized, refinement to the next level, $l+1$ then requires prolonging the current solution to this finer-resolution mesh, $x^t_{l+1} = P^l_{l+1}(x^*_l)$, after which we proceed by quasistatic stepping with solves of (3). However, while this prolonged geometry provides a natural starting point

[1] We apply Neo-Hookean membrane [Chen et al. 2018b] and discrete hinge bending [Grinspun et al. 2003; Tamstorf and Grinspun 2013] for shell elastics, and C-IPC [Li et al. 2021] barriers for contact, friction and strain limiting.

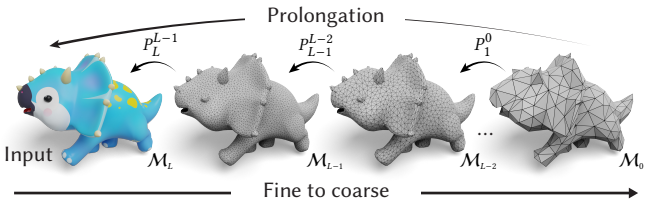


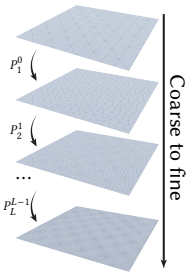
Fig. 5. **Fine-to-coarse construction of shell prolongation operators:** In contrast to PCS, which relies on a coarse-to-fine Loop subdivision process to construct a smooth mesh hierarchy, PSQ applies a fine-to-coarse hierarchy construction to support detailed irregular input meshes. We construct a non-linear prolongation operator P custom-designed for shell simulation during the forward mesh decimation process between each level. By repeatedly applying these prolongation operators, we can refine coarse shell geometry and deformations from the coarsest level-0 mesh, $\mathcal{M}_0$, through a sequence of meshes, $\mathcal{M}_l$, until reaching the finest level-L geometry, $\mathcal{M}_L$.

for the next level's solve, prolongation is oblivious to contact constraints and strain limits, and so can and will violate feasibility with both intersections and excessive stretching. As a final computation then, to start our progressive simulation solve at each new level, we must find a safe (intersection-free and strain-limit satisfying) initializer close to x^t_{l+1} in order to begin the new level's preview stepping with (3).

3.3 Progressive Cloth Simulation

Zhang et al.'s [2022] Progressive Cloth Simulation (PCS) constructs its hierarchy coarse-to-fine, starting with an input, coarse mesh triangulation, via a boundary-fixed Loop-subdivision [Loop 1987]. This forms a nested hierarchy of triangle meshes, with a *linear*, $P^l_{l+1} \in \mathbb{R}^{3n_{l+1} \times 3n_l}$, prolongation operator per level naturally defined by each level l's corresponding Loop subdivision operator. In turn direct prolongation from any level l to the finest scale is simply the repeated prolongation, $P^l(x_l) = P^{L-1}_L \cdots P^l_{l+1} x_l$.

For safe initialization PCS's nested hierarchy also provides a safe and simple, intersection-free initializer, x^s_{l+1}, via Barycentric upsampling followed by linear search along $d = P^l_{l+1}(x_l) - x^s_{l+1}$ to find a close-by feasible point to the prolongation. With an intersection-free point, any remaining strain-limit violations are then progressively removed by a stretch-reducing optimization drawing principal stretches below their limits.



4 PROGRESSIVE SIMULATION FOR SHELLS

Our goal is a progressive simulation suitable for both unstructured and structured triangle mesh simulation domains. In turn, this requires a corresponding prolongation operator and safe initialization, which a) captures both intrinsic (in-plane) and extrinsic (bending) deformation consistent with the shell models we simulate and b) preserves rest shapes at all levels in our hierarchy.

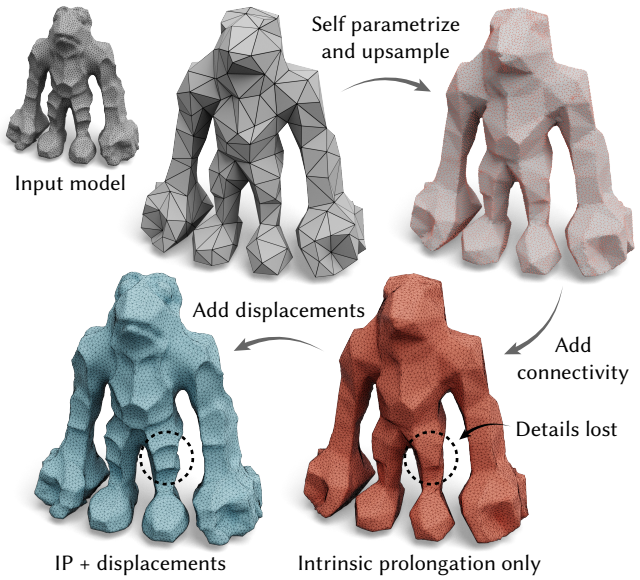


Fig. 6. **Shell Prolongation:** We can prolong coarse mesh vertices on our fine-to-coarse hierarchy with intrinsic parameterization. However, intrinsic parameterization does not recover the original rest shapes of the hierarchy – a requirement for progressive simulation. We extend intrinsic prolongation to capture missing extrinsic information with offset displacements defined per-shell element that correctly preserve detailed-geometry rest shapes, and help capture both intrinsic (stretch) and extrinsic (bend) deformations for shell simulation.

4.1 Simulation Hierarchy from Decimation

We begin hierarchy construction with an input, fine-resolution triangle mesh $\mathcal{M}_L$. This is the finest-level mesh we target for our progressive simulation's final, converged physical solution. We then build our simulation hierarchy by applying a series of edge-collapse decimations with subsequent levels' meshes, $\mathcal{M}_{L-1} \cdots \mathcal{M}_0$, each defined after intervals of decimation. For decimation, we apply quadric error edge collapse [Garland and Heckbert 1997] with probabilistic quadrics for high-quality decimation [Trettner and Kobbelt 2020]. We then apply each level l's initial decimation vertex positions to define that level's rest geometry, $\bar{x}_l$.

4.2 Prolonging Decimated Hierarchies

As reviewed and analyzed further in Section 5, recent methods offer initially promising opportunities for prolonging our edge-collapsed hierarchies with linear operators. While neither, unfortunately, are suitable for progressive shell simulation, they provide a good starting point to analyze necessary properties for shell prolongation, and to construct our prolongation operator.

Intrinsic prolongation [Liu et al. 2021], provides a sparse linear operator via repeated local self-parameterization. However, it does not preserve rest shapes in the hierarchy, generating significant stretch-response errors in all configurations, including ghost forces when the surface is undeformed. At the same time, it misses the critical bending responses of rest-shape curved surfaces. See Figure 6 and §5.

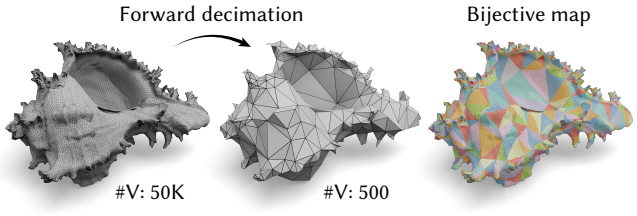


Fig. 7. **Mapping Shells:** In the forward decimation process, a bijective map is established between a detailed input shape and its simplified, coarsened version. This mapping allows us to select a specific location on the intricate mesh and directly find its corresponding position on the coarsened mesh and vice versa. We use the visualization method of [Liu et al. 2023] to show the color-coded, simplified mesh overlay onto the high-resolution model.

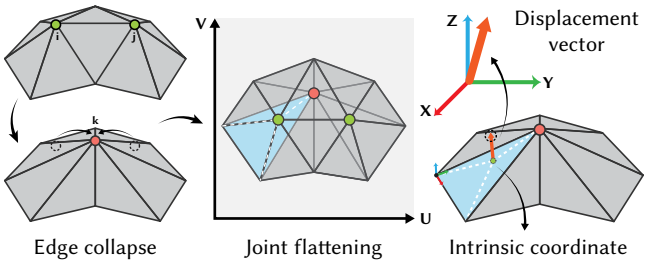


Fig. 8. **Edge-expansion quantities:** In the forward decimation process, we record the intrinsic coordinates of expanding edge vertices, i and j, in their local joint flattening. In turn, the offsets from these intrinsic coordinates to their rest vertex position are then also stored as corrective displacements in the triangle's local system.

On the other hand, affine prolongation [Liu and Jacobson 2019], via repeated best-fit affine transforms per edge-expansion neighborhood, nicely recovers the hierarchy's rest-shapes. However, this affine fitting incorrectly provides a volumetric approximation for a shell deformation (effectively approximating deformation per neighborhood with a linear tetrahedral element). In shell simulation, this generates poor force approximations during restriction, leading to jittering and stagnating convergence (see Figure 15). At the same time, this volumetric approximation is also often rank-deficient in many regions, leading to significant and unacceptable geometric artifacts in upsampling (see Figure 13). Finally, while similarly to intrinsic prolongation, affine prolongation also provides a *linear* operator, its lack of sparsity makes it impractical for high-resolution mesh simulation.

4.3 Shell Prolongation for Progressive Simulation

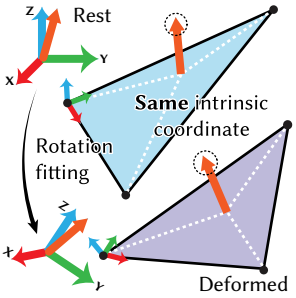
Target properties for our prolongation operator are then

(1) *Rest shape recovery:* Prolongation should preserve rest shapes at all levels of refinement in our hierarchy;
(2) *Shell-model consistency:* Prolongation should upsample both intrinsic (stretch) and extrinsic (bending) deformation modes of the codimensional shell geometries; and
(3) *Efficiency:* Progressive simulation applies repeated prolongation calls inside every force and refinement evaluation—it must be inexpensive to work within both operations.

We begin our construction by considering the simplest case of a hierarchy formed by a single-edge collapse (see Figure 8). The rest-shape prolongation that reverses this collapse is simply "edge expansion." We start with Intrinsic Prolongation's self-parameterization [Liu et al. 2021] to define the intrinsic (uv) coordinates that will anchor the locations of the two newly split vertices (for the next $l+1$ level mesh), $\bar{v}^i_{l+1}$ and $\bar{v}^j_{l+1}$, in the current mesh l.

We then store, one-time, the rest-shape extrinsic difference, $\bar{d}^i = \bar{x}^i_{l+1} - \bar{v}^i_{l+1}$, per new-level vertex i, with respect to the coarse level-l's rest-shape triangle containing $\bar{v}^i_{l+1}$ (Figure 8). With ordered vertex rest-positions, $\bar{x}_0, \bar{x}_1, \bar{x}_2$, for the containing triangle, we decompose the extrinsic difference into an out-of-plane offset $\gamma^i = \bar{n}^T \bar{d}^i \in \mathbb{R}$ along the triangle's normal, $\bar{n}$, and the remaining, in-plane contribution $t^i \in \mathbb{R}^2$, so that $\bar{d}^i = \bar{n}\gamma^i + (\bar{e}, \bar{n}\times\bar{e})t^i$, with $\bar{e} = \bar{x}_1 - \bar{x}_0/\|\bar{x}_1 - \bar{x}_0\|$.

During simulation, shell deformation drives the corresponding deformation of these stored in-plane and normal components. Our prolongation begins with the simple (and linear) intrinsic prolongation to determine the in-plane deformation of our anchoring point locations, v^i_{l+1}. Bending of our shell's hinge elements then determines each containing triangle's new normal direction, $n^i(x_l)$, for application of our out-of-plane contribution, γ^i. Finally, and correspondingly, the rotational factor[2], $R^i(F^i) \in \mathbb{R}^{3\times 2}$, of each containing triangle's deformation gradient, F^i, gives the rotation for our remaining, in-plane offset, t^i. See the inset for an illustration of this process.



Put together, prolongation during deformation for a single, edge-expanded vertex is then,

$$x^i_{l+1} = v^i_{l+1} + a^i(x_l), \tag{4}$$

where the nonlinear offset is

$$a^i(x_l) = n^i(x_l)\gamma^i + R^i(x_l)t^i, \tag{5}$$

and $(\gamma^i, t^i) \in \mathbb{R}^3$ are precomputed once.

Moving to our general case of prolongation with our hierarchies defined by many (often nested) edge-collapses then follows easily. Applying intrinsic prolongation to all our expanded nodes simultaneously to compute their intrinsic "anchor" locations remains a linear operation, $v_{l+1} = U^l_{l+1}x_l$. We then can similarly apply our nonlinear offset computation globally for all m_l expanded nodes as

$$a^l_{l+1}(x_l) = [a^1(x_l)^T, \cdots, a^{m_l}(x_l)^T]^T, \tag{6}$$

so that our per-level shell-prolongation operator is

$$x_{l+1} = P^l_{l+1}(x_l) = U^l_{l+1}x_l + a^l_{l+1}(x_l). \tag{7}$$

[2]With deformed positions x_0, x_1, x_2 and undeformed reference edges $\bar{e}_1 = \bar{x}_1 - \bar{x}_0$, $\bar{e}_2 = \bar{x}_2 - \bar{x}_0$, the membrane deformation gradient per containing triangle is $F(x) = (x_1 - x_0, x_2 - x_0)\bar{B}^{-1} \in \mathbb{R}^{3\times 2}$, where $\bar{B} = \begin{pmatrix} \|\bar{e}_1\| & (\bar{e}_1\cdot\bar{e}_2)/\|\bar{e}_1\| \\ 0 & \|\bar{e}_1\times\bar{e}_2\|/\|\bar{e}_1\| \end{pmatrix}$. Its rotation is then $R(F(x_l)) = U\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}V^T \in \mathbb{R}^{3\times 2}$ with the SVD $F(x_l) = U\Sigma V^T$.

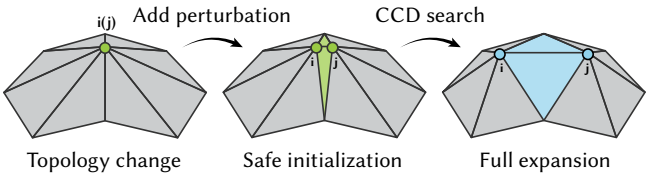


Fig. 9. **Safe Single Edge Expansion with Contact:** (Left) For a single edge expansion, we perform topology change, then (Middle) we carefully perform a safe collision-free initialization of edge positions via a robust randomized stratified sampling algorithm; and, once safely initialized, we (Right) apply our CCD-filtered geometric expansion until the full edge expansion is finalized.

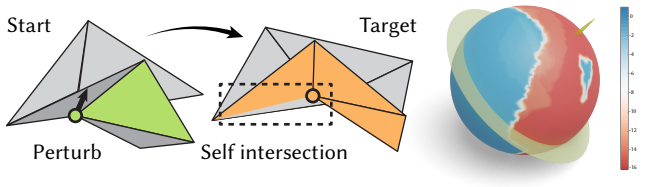


Fig. 10. **Safe Sampling:** Demonstration of the high likelihood of finding a safe perturbation direction for intersection-free initialization. Here, we consider an example input geometry in the top left, produced after PSQ's topology update step (see Figure 9), while the bottom left shows its corresponding target prolonged geometry with self-intersections. Each sampled point on the sphere (we use 10,000 random samples) represents a perturbation direction, with color indicating the maximum allowable step size (in log scale) along that direction with no self-intersection. Blue regions then denote safe perturbation regions, while the green plane aligns, for visualization, the directions aligning with the surface of the green triangle in the starting geometry.

4.3.1 Direct Prolongation. Along with our above, per-level prolongations, $P^l_{l+1}(\cdot)$, our mapping allows us to build direct prolongations for non-adjacent levels of our hierarchy. Specifically, to compose prolongations, $P^l(\cdot)$, that directly map from a level $l < L$ to our finest mesh, we apply the same above construction by precomputing and storing the extrinsic differences between the finest rest-mesh and its intrinsic prolongation from level l's rest mesh. We form the intrinsic prolongation from level l to L by repeated composition $U^l = U^{L-1}_L \cdots U^l_{l+1}$ and precompute our per-vertex displacements, (γ^i, t^i), via the difference between the resulting intrinsic prolongation $\bar{v} = U^l x_l$ and the *finest-level* rest geometry, $\bar{x}$. Once stored, at deformation all m edge-expanded vertices from level l to L are again prolonged via (4) so that

$$a^l(x_l) = [a^1(x_l)^T, \cdots, a^m(x_l)^T]^T, \tag{8}$$

and our direct prolongation operators is

$$x = P^l(x_l) = U^l x_l + a^l(x_l). \tag{9}$$

Note that for deformations away from rest, our direct prolongation is not equivalent to repeated application of our per-level prolongation, $P^l \neq P^{L-1}_L \circ \cdots \circ P^l_{l+1}$. While both operations preserve the finest-level rest shape, they obtain different (albeit similar) prolongations of the finest-resolution geometry under deformation. Direct prolongation, however, avoids the dense stencils and additional nonlinearity that repeated composition of single-level operators would introduce, improving efficiency in optimizing each level's progressive solution.

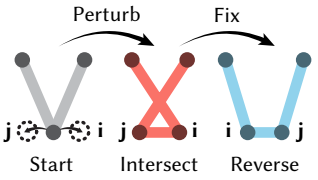
4.4 Safe Initialization for Refinement

As discussed in §3, the Progressive Simulation framework requires warm-starting our IPC-barrier-based solves (see §4.5 below for details) of each finer-level $l+1$'s minimization of (2) with a feasible (interpenetration-free and, when required, strain-limit satisfying) initialization close to the *prolongation* of the just-completed prior-level's solution, $P^l_{l+1}(x^{t+1}_l)$. This enables *safe*, progressive refinement of our equilibrium solution all the way from a feasible input geometry until final convergence at our finest-level mesh.

PCS [Zhang et al. 2022] builds a nested hierarchy via Loop subdivision and so can directly compute a contact-safe initialization to each finer-level's geometry by simply applying in-plane upsampling, followed by CCD-filtered advancement towards $P^l_{l+1}(x^{t+1}_l)$. However, to support arbitrarily structured mesh connectivities, and (non-nested) decimation-built hierarchies, this simple strategy is no longer applicable. We next address this challenge of computing feasible initializations for progressive simulation on arbitrary shell geometries and unstructured meshes (see Algorithm 2 in Appendix B for pseudocode.).

4.4.1 A safe initial expansion. As in our prolongation, our process for safe initialization in refinement applies edge expansion. Our first step is to build a safe and so intersection-free, edge-expanded non-degenerate starting mesh, x^s_{l+1}, with level l+1's topology, from which we can robustly search along the difference $d = P^l_{l+1}(x^{t+1}_l) - x^s_{l+1}$ to find a near-to-prolonged feasible point (see Figure 9).

We begin by examining a single edge-expansion patch in the refining mesh going from level l to $l+1$. An initial, purely topological expansion (no positional change) of an expanded vertex k, to vertices i and j, at vertex k's' location, generates an intersection-free geometry from level l's solution except for the coincident vertices.



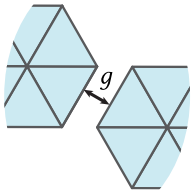
To apply a search for an intersection-free geometry from starting vertex positions of i and j, towards their target prolonged positions, we must first *safely* separate them by a small perturbation to enable robust distance calculations in the following CCD computations. In 2D, identifying a safe perturbation direction is simple: we can perturb towards the target prolonged positions of i and j, and in the reverse direction (see inset). However, without guidance, in 3D, even the smallest perturbation can easily generate severe local self-intersection in the patch (see Figure 10 Left) and even global intersections with other patches in the domain.

To find an initial safe perturbation per expansion, we propose a randomized stratified sampling algorithm as an ordered process to compute safe initializations for expansions. We start with a "first-attempt" perturbation that moves the edge-expanded vertices i and j towards their targets with small magnitudes guaranteed to avoid intersections outside the expansion patch. We then check perturbations for local self-intersection within the patch. If intersection-free,

we are done and can proceed to our CCD-filtered geometric expansion (see below). In the minority of cases, where self-intersection in the expansion patch is generated from this first perturbation, we next randomly sample perturbation directions (applying the same safe-magnitude bound to avoid global intersections) for vertex j and repeat our check until a safe initialization is obtained. While it is not possible to guarantee a safe perturbation exists for arbitrary geometric input (see our synthetic counter-example in Figure 20), empirically, we quickly obtain safe initialization from randomized sampling for all expansion examples across all our stress-test simulations. In Figure 10, we demonstrate the high likelihood of finding these safe perturbation directions for intersection-free initialization via random sampling. See Algorithm 5 in Appendix B for pseudocode.

4.4.2 CCD-filtered geometric expansion. Now that we have a method to perturb each edge expansion safely, we next detail our sequential process for safely expanding these perturbed edges towards their prolonged geometry targets via safe-stepping with CCD. Then, in the next section, we explain the final parallel safe-expansion algorithm this enables.

For CCD computation, we require finite edge lengths for robust distance evaluation; our above perturbation provides this. However, CCD efficiency (and speed of following time-step solves) drops as distances between mesh primitives become unnecessarily small. Our goal then becomes aggressively stepping expanded vertices towards their targets while ensuring applied displacements maintain a small, safe-distance gap between surface primitives throughout all expansions.



Before edge expansions are applied for refinement, the previous level's Newton-based IPC solution, x_l^{t+1}, ensures an easily evaluated minimal-gap distance is preserved between all surface primitives. We compute this gap, g, at the start of each refinement operation. The magnitude of this value remains large enough for efficient distance evaluations (here within the range of 10^{-4} to 10^{-3}m) by keeping the IPC contact barrier stiffness, κ, matched to each shell system's maximum bulk modulus [Li et al. 2021].

We then first displace each newly inserted vertex along the safe perturbation direction (determined by our above-described sampling method) by a distance that is an order of magnitude smaller than g. This ensures global non-intersection safety without requiring expensive, explicit global intersection checks and processing. Next, we apply conservative CCD-filtered steps to expand each new edge's vertices towards their targets geometrically. We apply Li et al.'s additive-CCD (ACCD) [Li et al. 2021] for all CCD evaluations. ACCD provides a conservative ratio bound, $r \in (0, 1)$, as an input parameter to each CCD evaluation between mesh-primitive pairs (point-triangle and edge-edge). Each such ACCD evaluation then returns a maximal step size that, when taken along the queried displacement direction, guarantees that the ratio between the new distance between the primitive pairs, d_1, and the initial distance, d_0, will be greater than r, so that $d_1 \geq r d_0$.

After culling far-away candidate pairs with standard collision-detection acceleration (spatial hash and broad-phase CCD), we first visit all remaining collision-candidate pairs (before ACCD) to choose, per query, a suitable ratio bound, r, that seeks to balance productively large step sizes for progress, against overly large steps, that would potentially bring mesh primitives too close together. When the initial distance for a pair is well below the gap, $d_0 \ll g$, we set $r = 0.999$ to minimize further decrease while not entirely preventing exploration via CCD of inward displacement for this pair altogether. On the other hand, when a pair's initial distance is of the same order or larger than the gap (i.e., $d_0 \approx g$ or $d_0 \gg g$), we balance progress with conservative exploration and set $r = 0.1$. See Algorithm 4 in Appendix B for pseudocode.

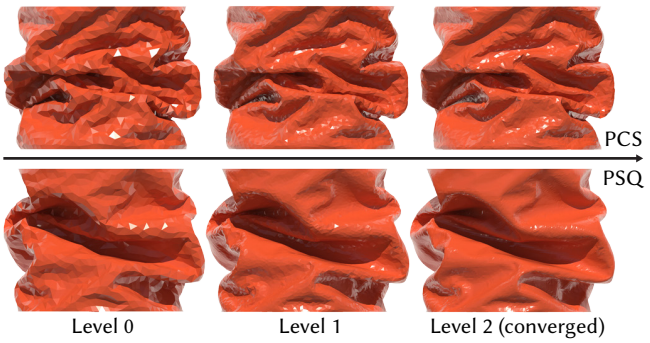


Fig. 11. **Crushing Results:** Mesh hierarchies generated by (prior work method) PCS on unstructured mesh input construct triangle meshes that are ill-suited for many shell simulations. Here, when we simulate the crushing of a thin-wall aluminum can, this causes PCS (Top Row) to introduce significant errors and locking artifacts. These artifacts are especially highlighted when compared to PSQ's results (Bottom row) for the same simulation, where both methods' hierarchies use the same base, level-0 unstructured mesh, and then refine with comparable DOF and triangle counts at each level.

4.4.3 Expansion-patch relaxation. We emphasize that the above-described ratio bounds for ACCD are heuristic choices that we find exceedingly effective for efficient gap maintenance in most of our refinement steps. However, the safe initialization process does not rely on these ratio choices. Instead, after applying the ACCD-determined maximal-step-size displacement, we check our final resulting distance per pair, d_1. If we find a too-small distance, $d_1 < g$, and/or triangle-element degeneracy (recalling that the CCD step is physics oblivious), we then locally relax the expansion patch with repeated Newton iterations of the incremental potential solve (3), with an increasing schedule of barrier stiffness, until the balance between elastic and IPC barrier forces restores the gap between the too-close collision pairs, and expands the ill-shaped elements. These per-patch relaxation solves are a small and local overhead that rarely exists in most steps due to our ACCD bound-ratio setting.

4.4.4 Safe parallel expansion. In the above section, for clarity, we introduce our safe edge-expansion process *sequentially*, visiting one edge expansion at a time. With the current local process detailed, we now describe here the fast, parallel variant that we apply for efficient, safe initialization for each refinement operation in our progressive simulation.

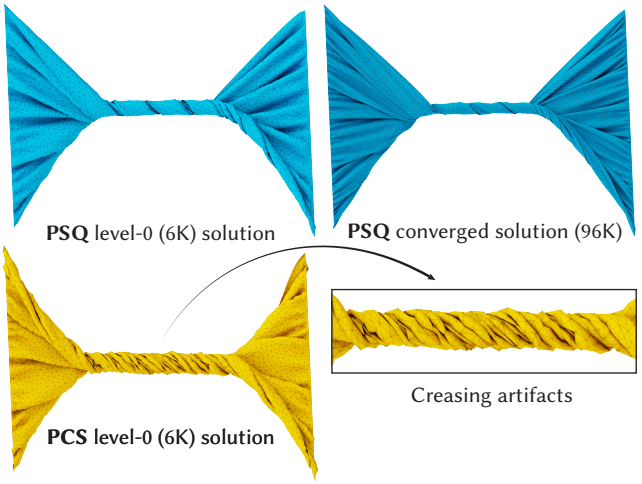


Fig. 12. **Lots of Twisting:** (Bottom row) progressive-simulation of a tightly twisting cloth with (prior work) PCS's Loop-subdivision-based method yields a coarse-level simulation with overly creasing spiral artifacts; whereas our PSQ solution remains artifact-free during coarse-level preview (top left), with tight conforming twists that are consistent with it's finest level, converged solution (top right).

For parallelization, a first naive strategy could be to insert and visit all edge expansions carrying us from mesh $\mathcal{M}_l$ to mesh $\mathcal{M}_{l+1}$ simultaneously. Unfortunately, this strategy already encounters a problem during the initial edge perturbation: parallel nested edge-expansions compromise the evaluated safe-gap bound and would, in turn, then require the solution of a challenging, global detangling problem with expensive global intersection queries and sampling.

Instead, we decompose the expansion process from $\mathcal{M}_l$ to $\mathcal{M}_{l+1}$ in parallelized batches via coloring. To start with, we require that vertices within each color can not be expanded simultaneously with a combinatorial neighbor. This defines independence for our coloring and ensures that no nested expansions are treated per batch. To build our graph coloring, we follow the dependency graph established by our original decimation to apply greedy coloring while maintaining the dynamically expanding mesh. This ensures that the order of any two dependent edge expansion operations cannot be reversed. As our graph coloring is then based solely on mesh *topology* we build it one-time as a precomputation, right after decimation, and then reuse it throughout our progressive simulation pipeline (see Figure 3 in the supplemental for visualization and Algorithms 6 and 7 in Appendix B for pseudocode.)

Applying our graph coloring, we process the safe edge expansions in parallel per batch. Within each batch, we start by parallel applying our above-described sampling-based safe perturbations. Next, for the same batch, we apply a global, additive advancement with our ACCD step that allows us to safely evaluate our CCD queries in parallel while still ensuring progress is not stalled early. After perturbation, each expanding vertex, x_i, in the batch has a corresponding remaining displacement to its target, δ_i. As above in §4.4.2 we visit, after collision culling, each remaining candidate pair to set their ACCD ratio bounds. Then, applying ACCD in parallel to all candidate pairs returns a maximal safe step size per pair. Safe progress then requires all vertices to advance via the smallest among these returned step sizes, α. If we stop here, as in the standard IPC line-search process, this unduly halts progress for all remaining vertices not in the stencils of the surface pairs that generated α. Instead, we additively repeat this process so that all vertices are productively stepped along their individual displacements.

To do so, inside each loop, we first query the collision-culled set for a new upper-bound allowed step size, α. We next safely advance *all* vertices i in the batch by $x_i \leftarrow x_i + \alpha\delta_i$. We then "stop" the advance of all vertices j that participate in a candidate-pair stencil, which generates the upper-bound step size α. We do this by canceling their remaining displacement, $\delta_j \leftarrow \mathbf{0}$. Finally, we update the displacement of all remaining vertices k in the batch to account for the applied advancement, $\delta_j \leftarrow (1-\alpha)\delta_j$. We repeat this loop until the parallel ACCD query returns $\alpha = 1$, indicating that no further progress toward the prolonged target is possible along the initial direction. Here, throughout this process, collision detection remains inexpensive as the broad-phase acceleration structure and its culling can be applied once at the start of the loop, which remains valid as all our queries are applied repeatedly on subsets of the same initial displacements.

Finally, as described above in §4.4.3, if any primitive pairs end this process with a distance less than our safe gap, g, we apply our patch-based relaxation, now solving over local patches in the current color's domain until the elastic and IPC barrier forces restore the gap and correct ill-shaped elements. See Algorithm 3 in Appendix B for pseudocode.

4.4.5 Strain Limiting Feasibility. Our safe parallel edge-expansion gives us an interpenetration-free target mesh, at the new level $l+1$'s resolution, near to our prolonged target, $P_{l+1}^{l}(x_l^{t+1})$. However, the newly defined mesh can still violate the membrane deformation limits imposed for strain-limited shell materials. To apply a strain-limit correction, we then observe that we satisfy all necessary conditions to apply the PCS method's variational strain-limit correction solve [Zhang et al. 2022]. To do so, in direct analogy to PCS's treatment of even/odd vertices in its Loop subdivision, we simply fix pre-existing vertices from our prior coarse level mesh, $\mathcal{M}_l$, and iteratively optimize our newly inserted vertices from $\mathcal{M}_{l+1}$ to enforce strain limits at the new level. Our strain-limit correction then remains otherwise unchanged from PCS [Zhang et al. 2022].

4.5 Quasistatic Stepping

Once we have computed the safe initializer to start the next level's refinement, we begin solving the new level's preview. We repeatedly solve (3) with Newton-type iterations to step our preview solutions. To do so, we begin with Zhang et al.'s [2022] inexact Newton strategy – using the projected Hessian of each coarse-level's incremental potential, $H(x_l) = \frac{1}{h^2}M_l + \text{ProjectPSD}\big(\nabla^2 E(x_l)\big)$, for preconditioning. Then, in each Newton-type iteration, with the current estimated solution, x_l, we further approximate the gradient by fixing the prolongation's current offset to the currently computed displacement, $d_l = a^l(x_l)$. Each iteration's search direction, p, is then obtained by

solving the linear system

$$H(x_l)\, p = -\big(\frac{1}{h^2}M + (U^l)^T \nabla\Psi(U^l x_l + d_l) + \nabla C(x_l)\big). \qquad (10)$$

For progressive simulation, each individual quasistatic step need not be solved accurately. Instead, we apply a small maximum number of solution-improving iterations per proxy step and quickly progress the preview solution with both rapid convergence to equilibria and stable, easily interruptible, intermediate previews during refinement. For all preview levels $l < L$, we stop quasistatic stepping when the current change in solution drops below user-specified tolerance, $||\alpha p|| < \epsilon$ where α is the returned step size from line search. After the last refinement is completed, our stepper directly solves the full system's incremental potential (1) again via quasistatic stepping for the finest level to convergence. See Algorithm 1 in Appendix B for pseudocode.

4.6 Progressive Collision Objects

Along with enabling the simulation of shell elastics, the combined application of our prolongation and safe initialization now also enables direct integration of arbitrary collision-object geometries (both scripted and fixed) into the progressive simulation pipeline. *Progressive collision objects* can now simply be included without requiring special-case treatment or mesh restrictions at input along with all other triangle mesh domains. They are initialized and pre-processed in the hierarchy, prolongation, and coloring construction identically to all the other input meshed domains. Then, at each level of simulation, we simply record the per-step target positions for each collision-object node. During that level's quasi-static simulation, vertices in the collision-object domains are equipped with IPC's augmented-Lagrangian energy [Li et al. 2020a], driven to their targets during each solve iteration. Collision objects are then treated identically to the physical shell domain for all other steps of the progressives-simulation pipeline.

5 EVALUATION

We implement our methods in C++, applying PARDISO [Bollhöfer et al. 2020], compiled with Intel MKL LAPACK and BLAS for linear solves and Eigen for remaining linear algebra routines [Guennebaud et al. 2010]. As covered above for robust conservative continuous collision detection we evaluate queries with spatial-hash culled ACCD [Li et al. 2021]. We summarize example statistics in the supplemental and report timings with a M1 Max (32 GB) MacBook.

5.1 Comparisons

PCS's Loop Subdivision Prolongation. While our primary focus is on enabling the progressive simulation of curved shell surfaces with unstructured meshes, our PSQ simulation works equally well with flat rest-shape geometries. In such cases, our nonlinear prolongation operator simplifies to linear barycentric interpolation. In draping simulations with flat cloth sheets, our coarse-mesh previewing model then aims to approximate fine-mesh curvatures with fewer triangles. In Figure 2 in the supplemental we see a significant difference even in a simplest case of flat-sheet draping. Here, while both the PCS and PSQ preview solutions well-capture their final respective fold arrangements, PCS's Loop subdivision prolongation significantly over-estimates the overall material stiffness resulting in a cloth silhouette that "droops" much more in its final highest-resolution, converged solution, whereas the PSQ solution significantly improves in capturing the material's bulk stiffness in its preview.

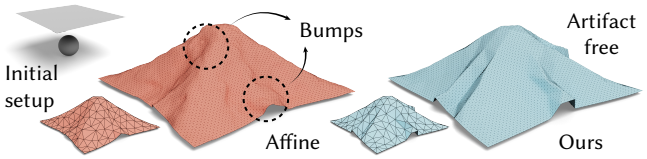


Fig. 13. **The Bumps:** Affine prolongation is often rank-deficient in many regions, leading to significant and unacceptable geometric artifacts in upsampling like the bumps we see here (Left). In contrast PSQ's prolongation (Right) recovers smooth, artifact-free prolongations that offer the next stage of optimization a smooth ride towards convergence.

More generally, mesh hierarchies that are generated by PCS via its repeated Loop subdivision on unstructured mesh input, generate poor triangle meshes for both cloth and shell simulation that can introduce significant artifacts when compared to our PSQ method's hierarchy. In Figures 12 and 11 we set up two very different stress-tests comparing the results of PCS and PSQ. In these simulations, both of the two methods' hierarchies use the same base, level-0, unstructured mesh, and then refine with comparable DOF and triangle counts at each level. In Figure 12 we twist a one meter square cloth, incrementally rotating its ends for 15s ($h = 0.1s$). As the simulation progresses we see that the PCS coarse-level preview-simulations yield tight, overwound and creased spirals (creasing artifacts) significantly different than the final-level solution, while the PSQ solution remains consistent and artifact free during coarse-level preview, despite the tight conforming twists, all the way out to it's finest level result (see Figure 3). Next in Figure 11 we switch to a thin-wall (0.01mm) 10cm tall aluminum cylinder that we incrementally crush. Here we see that the resulting PCS solutions suffer from significant artificial locking resulting in poor and jagged solutions at both levels' of its preview hierarchy, as well as in its final converged solution. Again, in contrast, we see that despite the exceedingly thin and stiff material, our PSQ previews capture the complex initial buckling behavior, even on our coarsest mesh, with consistent and refined results up to the converged solution (Figure 4). Likewise in Figure 4 we also see that a direct coarse-level simulation suffers even-more significantly from comparable membrane-locking issues – please again contrast with the coarse-level PSQ solution. Here these results emphasize how PSQ significantly improves shell modeling in both coarse-level simulation and final-level results by 1) enabling the use of user-provided high-quality, finest-level meshes (not possible for PCS) for force and energy evaluations; and 2) replacing PCS's sub-optimal choice of the Loop-subdivision operator for prolongation with PSQ's shell-model-customized prolongation.

Intrinsic Prolongation. As covered in §4, intrinsic prolongation provides a linear, sparse option for prolongation by tracking a bijective mapping between local patches during edge applications. This works well when the aim is to map surface signals between

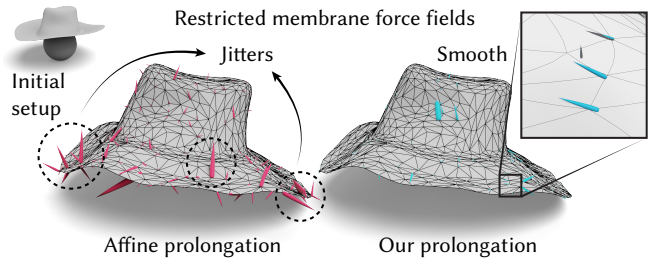


Fig. 14. **Affine Jitters:** We compare the stretch forces generated by affine prolongation to those generated by our prolongation operator. In simulation, affine prolongation generates large jittering artifacts which stymie convergence. We see that reflected in the affine solution's noisy force field. Conversely, our prolongation smoothly diffuses the force field leading to convergent solutions at all levels of the hierarchy.

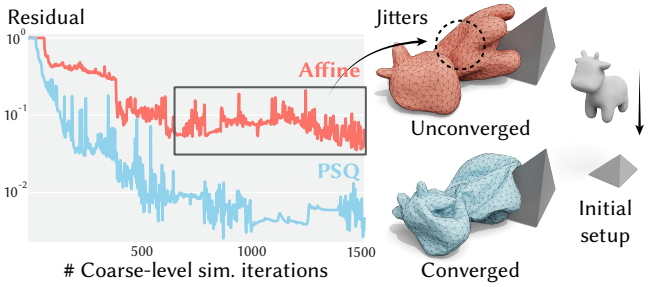


Fig. 15. **Jittery Convergence:** Due to errors in its force projection, the affine operator generates jittering artifacts and stagnated convergence that halts solutions well away from equilibrium (Top-Right). In contrast our PSQ prolongation converges to low tolerance solutions that capture detailed wrinkling deformation (Bottom-Right), without artifact.

different levels for linear multigrid solving. However, when applied directly to progressive simulation evaluations, this results in the loss of extrinsic surface details relative to the original input and so the inability to capture rest-shape in simulation (see Figure 6).

Affine Progressive Meshes. Best-fit affine transformation [Liu and Jacobson 2019], can be applied as a rest-shape preserving, linear prolongation operator. However, there are three practical limitations that make it unusable for progressive simulation:

- *Lack of sparsity:* As the number of edge decimations increases, the prolongation matrix created via the affine prolongation becomes prohibitively dense (please see our discussion in Appendix A).
- *Prolongation artifacts:* For flat regions the rank deficiency in the affine fitting (despite the remedy provided by Tikhonov regularization) produces significant artifacts, making the prolonged geometry unsuitable for visualization and force sampling (see Figure 13).
- *Force restriction artifacts:* Most importantly, affine prolongation performs poorly in restriction, i.e., when downsampling forces, which leads to jittering artifacts and so stagnated convergence; compare to our prolongation's results in Figure 15.

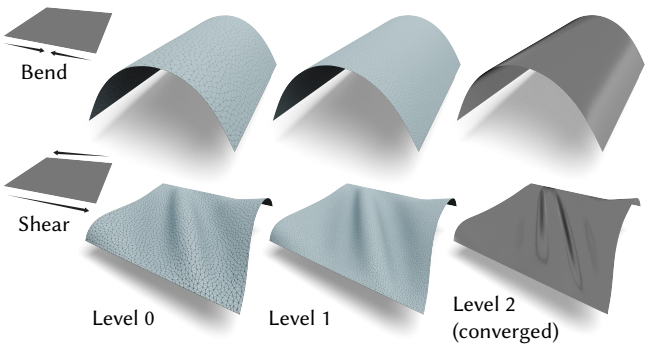


Fig. 16. **Plate Tests:** Bending and shearing "unit tests" with a stiff metal sheet.

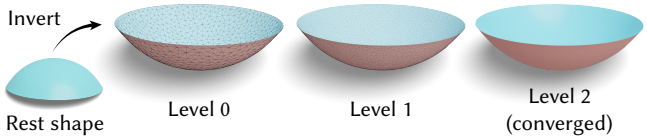


Fig. 17. **Shell Test:** Inversion "unit test" with a stiff metal dome.

Direct Fine and Coarse Resolution Simulation. Along with providing rapid and consistent preview and exploration across parameter changes, PSQ also offers significantly faster simulation of final high-resolution results over direct simulation. To get a sense of this speedup we now compare performance on the above-described cloth-twist and can-crush examples. For each example we apply the same incremental twisting/loading to the a fine-level direct simulation, coarse-level direct simulation and PSQ. For the crushed can PSQ's coarse preview simulation to equilibrium takes 9.36s incurring only a slight overhead in comparison to the coarse direct simulation (7.18s) while entirely avoiding the extreme membrane locking artifacts exhibited by the coarse simulation (Figure 4). On the other hand PSQ obtains 28X speedup over direct simulation to reach its final converged high-resolution simulation result (PSQ: 104.8s with a breakdown of 40.4s for equilibrium solves and 64.4s for safe initializations, Direct: 2987.8s). For the coarse level twisting we again see comparable timings to equilibrium (long twist loading time) for direct coarse and PSQ coarse preview (PSQ: 160.2s, Direct: 178s), despite significant artifacts in the coarse direct simulation, while PSQ now obtains a 4X speedup over direct simulation to reach its final converged high-resolution result (PSQ: 548.2s with equilibrium solves 200.0s and safe initializations 348.2s, Direct: 2000.2s).

5.2 Tests and Examples

Shell "unit tests". We start by examining a set of basic shell and plate "unit tests" for stiff materials to confirm consistency in buckling. *Plate buckling:* we evenly press two sides of a square metal plate inwards; two out-of-plane stable solutions (buckling up or down) are possible; in Fig 16 (top) we see that we obtain the same equilibria at all level's of the PSQ hierarchy. *Plate shearing:* we evenly translate two sides of a square metal plate in opposing directions; the resulting large strain enables a wide range of possible stable states;

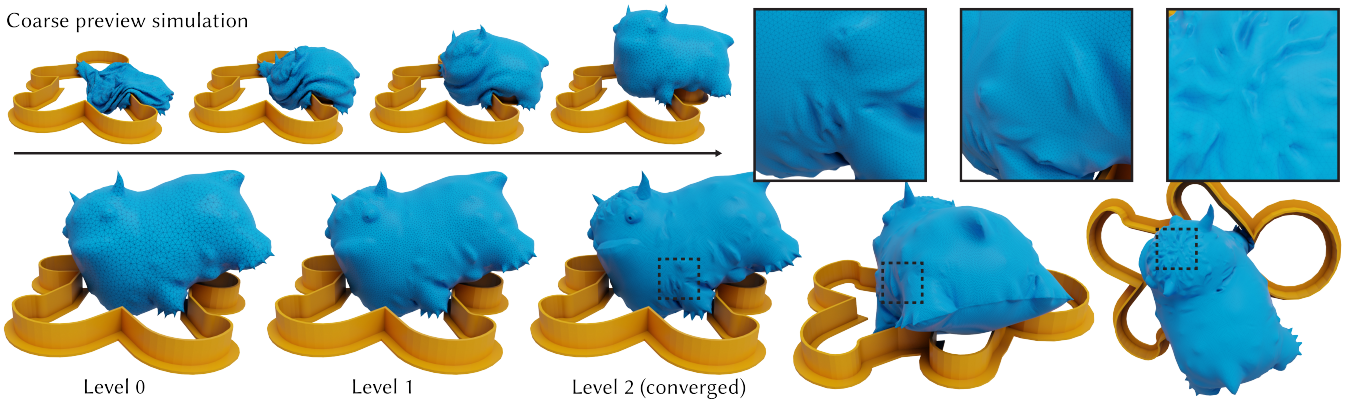


Fig. 18. **Tiny Inflatable Monster Seeking Cookies Progressively:** (Top-Left) During a designer's coarse-simulation-preview phase, a hollow shell of a monster is accidentally dropped onto a rigid cookie cutter, then gradually inflated a respectable amount, and re-posed. Next, during a progressive refinement phase (Bottom, Left-to-Right) the detailed contact-induced character deformations rapidly emerge using PSQ. The resulting converged simulation (Bottom-Middle) is faithfully represented by the coarse proxy, but includes (Right) interesting artifact-free shell deformation details. The entire process would be orders-of-magnitude slower and non-interactive using only fine-scale methods. Our method guarantees that the monster did not suffer any interpenetrations.

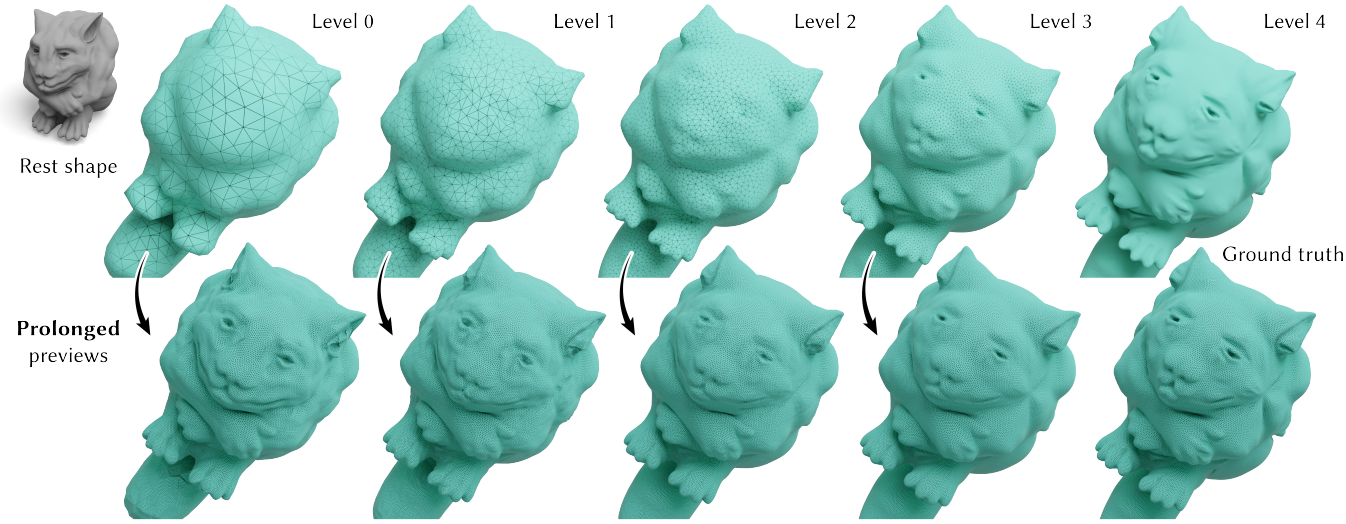


Fig. 19. **Prolonged Attention:** PSQ's prolongation operator applied to coarser-level mesh results (top row) on its own already provides a significant, high-quality preview for upsampled simulation geometry (bottom row).

in Fig 16 (bottom) we see PSQ obtain the same sheared pattern at all level's of the PSQ hierarchy. *Shell indent:* in Figure 17 we indent a thin acrylic shell through to what should be a second, equally stable inverted shape; here we see the PSQ solution consistently remains in the indented shape and does not snap through.

Crusher: While cylinder buckling is well-studied, predicting the bucking behavior of non-developable surfaces generally requires significant simulation overhead. In Figure 2 we crush a snarky aluminum pumpkin (0.01mm wall thickness, 10cm high) with a press, obtaining complex buckle patterning that only succeeds in making our pumpkin grumpier.

Balloons: Predicting the final shape and compliance of an inflated shape remains critical for soft robotics, orbital structures, birthday parties, and parades. In Figures 1, 18, and 19 we deploy deflated character balloon shapes – some resting on collision shapes and then progressively refine them with PSQ, after inflation, to their high-resolution shapes. Coarse-mesh PSQ previews across simulation scales, well capture the fine-scale shapes of their final, converged solutions, complete with the deformed characters' detailed geometries and wrinkling. In Figure 4 in the supplemental our bubble-bee balloon has a surprise inside – an inverted bubble cavity that only makes its appearance during PSQ's coarse-level simulation when sufficient pressure unfolds it from within and deploys it outwards.

PSQ's coarse-level prediction of this bubble's equilibrium shape then closely matches the final, fine-level simulation result.

6 CONCLUSION

We have presented Progressive Shell Quasistatics (PSQ), a new method that extends progressive simulation to the high-fidelity modeling and design of all input thin shell (and plate) geometries with unstructured (as well as structured) triangle meshes. PSQ generalizes prior work on progressive cloth simulation (PCS) [Zhang et al. 2022] by supporting fine-to-coarse hierarchies and so without the prior limitations restricting progressive simulation to work solely with subdivision connectivity meshes. Despite the wide generality of PSQ in its support for plates, shells, and arbitrary collision objects, there remains interesting future work to explore, and limitations to address. As with PCS, we observe good qualitative and quantitative consistency across scales for PSQ; however, there is no guarantee that this will hold for arbitrary shell model problems. For example, while buckling phenomena and equilibria have been faithfully reproduced with coarse-scale proxy models in our evaluation examples, there is no guarantee that arbitrarily coarse proxy models will be sufficient to estimate a particular targeted shell behavior. Exploration of the suitable coarsest resolution, per selected progressive simulation setting, is thus an important future direction to investigate. Similarly, our safe initialization method for refinement has supported our full range of practical and highly challenging, high-contact shell-simulation scenarios. However, despite this, as demonstrated in our pathological synthetic counterexample in Figure 20, there can be no guarantee that it will always succeed. Generalizing to a practical, safe initialization method with guarantees remains an open problem.

The mechanical modeling of shell structures in many domains, such as automotive and aerospace applications, can involve complex shell structures that may demand further enriched simulation models. For instance, many inter-connected and bonded parts may be difficult to model via only manifold-with-boundary meshes, e.g., the welded T-junctions, struts, and ribs used to model shell reinforcements. These and other applications require interesting extensions to resolve inter-shell constraints and structures in the progressive simulation of non-manifold shells.

ACKNOWLEDGMENTS

Jiayi Eris Zhang is supported by a Stanford Graduate Fellowship. Alec Jacobson is funded in part by the Canada Research Chairs Program and a Sloan Fellowship. We thank Hsueh-Ti Derek Liu for insightful discussion and help in making Figure 18.

REFERENCES

Burak Aksoylu, Andrei Khodakovsky, and Peter Schröder. 2005. Multilevel solvers for unstructured surface meshes. *SIAM Journal on Scientific Computing* 26, 4 (2005), 1146–1165.

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

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

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

Jan Bender, Daniel Weber, and Raphael Diziol. 2013. Fast and stable cloth simulation based on multi-resolution shape matching. *Computers & Graphics* 37, 8 (2013), 945–954.

Matthias Bollhöfer, Olaf Schenk, Radim Janalik, Steve Hamm, and Kiran Gullapalli. 2020. State-of-the-art sparse direct solvers. In *Parallel Algorithms in Computational Science and Engineering*. Springer, 3–33.

Mario Botsch and Leif Kobbelt. 2003. Multiresolution surface representation based on displacement volumes. In *Computer Graphics Forum*, Vol. 22. Wiley Online Library, 483–491.

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. *ACM Trans. on Graph.* 21 (05 2002).

Saifon Chaturantabut and Danny C. Sorensen. 2010. Nonlinear Model Reduction via Discrete Empirical Interpolation. *SIAM Journal on Scientific Computing* 32, 5 (2010), 2737–2764. https://doi.org/10.1137/090766498

Hsiao-Yu Chen, Arnav Sastry, Wim M van Rees, and Etienne Vouga. 2018b. Physical simulation of environmentally induced thin shell deformation. *ACM Trans. Graph. (TOG)* 37, 4 (2018), 1–13.

Jiong Chen, Hujun Bao, Tianyu Wang, Mathieu Desbrun, and Jin Huang. 2018a. Numerical coarsening using discontinuous shape functions. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–12.

David Clyde, Joseph Teran, and Rasmus Tamstorf. 2017. Modeling and data-driven parameter estimation for woven fabrics. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 1–11.

Gilles Daviet. 2020. Simple and scalable frictional contacts for thin nodal objects. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 61–1.

Fernando De Goes, Mathieu Desbrun, Mark Meyer, and Tony DeRose. 2016. Subdivision exterior calculus for geometry processing. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–11.

Marvelous Designer. 2022. https://www.marvelousdesigner.com

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 209–216.

Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. 2007. Efficient simulation of inextensible cloth. In *ACM SIGGRAPH 2007 papers*. 49–es.

Eitan Grinspun, Anil N Hirani, Mathieu Desbrun, and Peter Schröder. 2003. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Citeseer, 62–67.

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, Benoît Jacob, et al. 2010. Eigen v3.

Qi Guo, Xuchen Han, Chuyuan Fu, Theodore Gast, Rasmus Tamstorf, and Joseph Teran. 2018. A material point method for thin shells with frictional contact. *ACM Trans. Graph. (TOG)* 37, 4 (2018), 1–15.

Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. 2000. Normal meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 95–102.

David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. 2009. Asynchronous contact mechanics. In *ACM Trans. on Graph. (TOG)*, Vol. 28. ACM.

David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. 2008. Robust treatment of simultaneous collisions. In *ACM SIGGRAPH 2008 papers*. 1–4.

Hugues Hoppe. 1996. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 99–108.

Doug L James and Dinesh K Pai. 2003. Multiresolution green's function methods for interactive simulation of large-scale elastostatic objects. *ACM Transactions on Graphics (TOG)* 22, 1 (2003), 47–82.

Chenfanfu Jiang, Theodore Gast, and Joseph Teran. 2017. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Trans. Graph. (TOG)* 36, 4 (2017).

Lily Kharevych, Patrick Mullen, Houman Owhadi, and Mathieu Desbrun. 2009. Numerical coarsening of inhomogeneous elastic materials. *ACM Transactions on graphics (TOG)* 28, 3 (2009), 1–8.

Theodore Kim. 2020. A Finite Element Formulation of Baraff-Witkin cloth. In *Symposium on Computer Animation*.

Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 105–114.

Leif Kobbelt, Jens Vorsatz, and Hans-Peter Seidel. 1999. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry* 14, 1-3 (1999), 5–24.

Petr Krysl, Sanjay Lall, and Jerrold E Marsden. 2001. Dimensional model reduction in non-linear finite element dynamics of solids and structures. *International Journal for numerical methods in engineering* 51, 4 (2001), 479–504.

Aaron Lee, Henry Moreton, and Hugues Hoppe. 2000. Displaced subdivision surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 85–94.
Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence C. Cowsar, and David P. Dobkin. 1998. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, Orlando, FL, USA, July 19-24, 1998*, Steve Cunningham, Walt Bransford, and Michael F. Cohen (Eds.). ACM, 95–104. https://doi.org/10.1145/280814.280828
Cheng Li, Min Tang, Ruofeng Tong, Ming Cai, Jieyi Zhao, and Dinesh Manocha. 2020b. P-Cloth: Interactive Cloth Simulation on Multi-GPU Systems using Dynamic Matrix Assembly and Pipelined Implicit Integrators. *ACM Transaction on Graphics (Proceedings of SIGGRAPH Asia)* 39, 6 (December 2020), 180:1–15.
Jie Li, Gilles Daviet, Rahul Narain, Florence Bertails-Descoubes, Matthew Overby, George E Brown, and Laurence Boissieux. 2018. An implicit frictional contact solver for adaptive cloth simulation. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–15.
Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020a. Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics. *ACM Trans. Graph.* 39, 4 (2020).
Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021. Codimensional Incremental Potential Contact. *ACM Trans. Graph.* 40, 4, Article 170 (jul 2021), 24 pages.
Hsueh-Ti Derek Liu, Mark Gillespie, Benjamin Chislett, Nicholas Sharp, Alec Jacobson, and Keenan Crane. 2023. Surface Simplification using Intrinsic Error Metrics. arXiv:2305.06410 [cs.GR]
Hsueh-Ti Derek Liu and Alec Jacobson. 2019. Cubic Stylization. *ACM Transactions on Graphics* (2019).
Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. 2021. Surface Multigrid via Intrinsic Prolongation. *ACM Trans. Graph.* 40, 4 (2021).
Charles Loop. 1987. *Smooth Subdivision Surfaces Based on Triangles.* Master's thesis. University of Utah.
Mickaël Ly, Jean Jouve, Laurence Boissieux, and Florence Bertails-Descoubes. 2020. Projective dynamics with dry frictional contact. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 57–1.
Josiah Manson and Scott Schaefer. 2011. Hierarchical Deformation of Locally Rigid Meshes. *Computer Graphics Forum* (2011). https://doi.org/10.1111/j.1467-8659.2011.02074.x
Eder Miguel, Derek Bradley, Bernhard Thomaszewski, Bernd Bickel, Wojciech Matusik, Miguel A Otaduy, and Steve Marschner. 2012. Data-driven estimation of cloth simulation models. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library.
Rahul Narain, Tobias Pfaff, and James F. O'Brien. 2013. Folding and Crumpling Adaptive Sheets. *ACM Trans. Graph.* 32, 4, Article 51 (jul 2013), 8 pages.
Rahul Narain, Armin Samii, and James F. O'Brien. 2012. Adaptive Anisotropic Remeshing for Cloth Simulation. *ACM Trans. Graph.* 31, 6, Article 152 (nov 2012), 10 pages.
Matthieu Nesme, Paul G Kry, Lenka Jeřábková, and François Faure. 2009. Preserving topology and elasticity for embedded deformable models. In *ACM SIGGRAPH 2009 papers*. 1–9.
Miguel Otaduy, Rasmus Tamstorf, Denis Steinemann, and Markus Gross. 2009. Implicit Contact Handling for Deformable Objects. *Comp. Graph. Forum* 28 (04 2009).
Nikolas Schmitt, Martin Knuth, Jan Bender, and Arjan Kuijper. 2013. Multilevel Cloth Simulation using GPU Surface Sampling. *VRIPHYS* 13 (2013), 1–10.
Christian Schumacher, Bernd Bickel, Jan Rys, Steve Marschner, Chiara Daraio, and Markus Gross. 2015. Microstructures to control elasticity in 3D printing. *ACM Transactions on Graphics (Tog)* 34, 4 (2015), 1–13.
Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. 2008. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE transactions on visualization and computer graphics* 15, 2 (2008), 339–350.
SideFX. 2022. *Houdini Vellum.* https://www.sidefx.com/products/houdini/
Rasmus Tamstorf and Eitan Grinspun. 2013. Discrete bending forces and their Jacobians. *Graphical models* 75, 6 (2013), 362–370.
Rasmus Tamstorf, Toby Jones, and Stephen F McCormick. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph. (TOG)* 34, 6 (2015), 1–13.
Min Tang, Ruofeng Tong, Rahul Narain, Chang Meng, and Dinesh Manocha. 2013. A GPU-based streaming algorithm for high-resolution cloth simulation. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 21–30.
Min Tang, Huamin Wang, Le Tang, Ruofeng Tong, and Dinesh Manocha. 2016. CAMA: Contact-aware matrix assembly with unified collision handling for GPU-based cloth simulation. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 511–521.
Min Tang, Tongtong Wang, Zhongyuan Liu, Ruofeng Tong, and Dinesh Manocha. 2018. I-Cloth: Incremental Collision Handling for GPU-Based Interactive Cloth Simulation. *ACM Transaction on Graphics (Proceedings of SIGGRAPH Asia)* 37, 6 (November 2018), 204:1–10.
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*. 205–214.
Philip Trettner and Leif Kobbelt. 2020. Fast and Robust QEF Minimization using Probabilistic Quadrics. *Computer Graphics Forum* (2020). https://doi.org/10.1111/cgf.13933
Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. 2011. Sensitive Couture for Interactive Garment Modeling and Editing. *ACM Trans. Graph.* 30, 4, Article 90 (jul 2011), 12 pages.
Pascal Volino and N Magnenat Thalmann. 2000. Implementing fast cloth simulation with collision response. In *Proceedings Computer Graphics International 2000*. IEEE.
Etienne Vouga, David Harmon, Rasmus Tamstorf, and Eitan Grinspun. 2011. Asynchronous variational contact mechanics. *CMAME* 200, 25-28 (2011).
Huamin Wang. 2021. GPU-Based Simulation of Cloth Wrinkles at Submillimeter Levels. *ACM Trans. Graph.* 40, 4, Article 169 (jul 2021), 14 pages.
Zhendong Wang, Longhua Wu, Marco Fratarcangeli, Min Tang, and Huamin Wang. 2018. Parallel Multigrid for Nonlinear Cloth Simulation. *Computer Graphics Forum* (2018).
Clarisse Weischedel. 2012. A discrete geometric view on shear-deformable shell models. (2012).
Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A Scalable Galerkin Multigrid Method for Real-time Simulation of Deformable Objects. *ACM Trans. Graph. (TOG)* 38, 6 (2019).
Juyong Zhang, Yue Peng, Wenqing Ouyang, and Bailin Deng. 2019. Accelerating ADMM for Efficient Simulation and Optimization. *ACM Trans. Graph.* 38, 6, Article 163 (nov 2019), 21 pages.
Jiayi Eris Zhang, Jérémie Dumas, Yun (Raymond) Fei, Alec Jacobson, Doug L. James, and Danny M. Kaufman. 2022. Progressive Simulation for Cloth Quasistatics. *ACM Trans. Graph.* 41, 6, Article 218 (nov 2022), 16 pages. https://doi.org/10.1145/3550454.3555510
Denis Zorin, Peter Schröder, T Derose, L Kobbelt, A Levin, and W Sweldens. 2000. Subdivision for modeling and animation. SIGGRAPH 2000 Course Notes. *Organizers: Zorin, D., Schröder, P* (2000).
Denis Zorin, Peter Schröder, and Wim Sweldens. 1997. Interactive multiresolution mesh editing. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 259–268.

A 2D RESTRICTION ANALYSIS

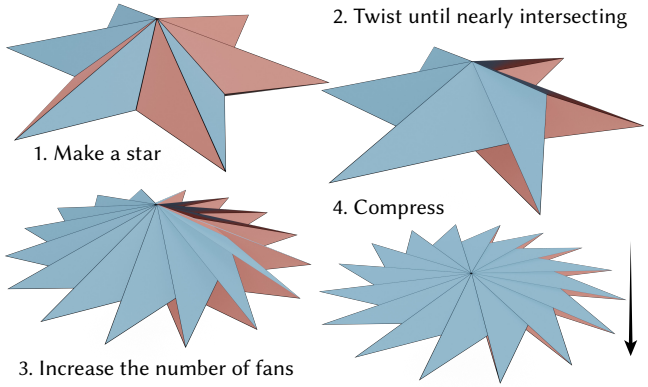


Fig. 20. **Recipe for cooking up a counter-example for our randomized safe-direction sampling:** 1. Make a 3D star shape. 2. Twist it nearly to self-intersection. 3. Increase the number of sides on the star. 4. Flatten the height of the star.

Here, we examine the application of affine mapping [Liu and Jacobson 2019] for restriction and demonstrate Intrinsic Prolongation's (IP) significant improvement over affine prolongation for downsampling forces. Consider the simple 2D example in Figure 21 with a fine polyline mesh in blue. After edge-collapsing the center line, we produce the coarse mesh given in red. Vertices across both meshes are sequentially numbered from 0 to n ($n = 2$ or 3), proceeding from left to right. To reiterate, our goal is a linear map P to transform signals living on the coarse mesh to the fine mesh. We define coarse vertices at rest $(\bar{x}_{c0}, \bar{x}_{c1}, \bar{x}_{c2})$ and deformed (x_{c0}, x_{c1}, x_{c2}). Similarly, rest and deformed fine vertices are $(\bar{x}_{f0}, \bar{x}_{f1}, \bar{x}_{f2}, \bar{x}_{f3})$ and

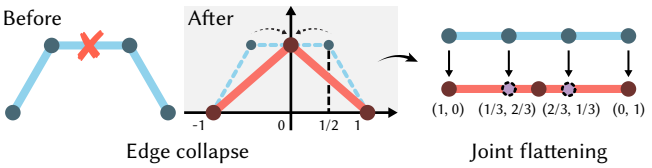


Fig. 21. 2D polyline analysis for restriction. Blue polylines define a 2D "shell" fine mesh. A single edge-collapse of the middle line (joining the endpoints and linking the residual points) gives the coarse mesh in red. Intrinsic parameterization concurrently flattens the two polyline meshes onto the same 1D domain to derive their intrinsic coordinates.

$(x_{f0}, x_{f1}, x_{f2}, x_{f3})$, respectively. Coarse rest and deformed bases are $\bar{B} = (\bar{x}_{c0} - \bar{x}_{c1}, \bar{x}_{c2} - \bar{x}_{c1})$ and $B = (x_{c0} - x_{c1}, x_{c2} - x_{c1})$.

At the coarse level, the affine mapping is defined for each fine vertex $x_{fi}, i \in [0, 3]$:

$$x_{fi} - x_{c1} = A(\bar{x}_{fs} - \bar{x}_{c1}),$$

where A is the least-squares affine-fit matrix with $A = B\bar{B}^{-1}$ ($B = A\bar{B}$ and $\bar{B}$ are invertible). Equivalently, we have barycentric coordinates β^i for each fine vertex i via

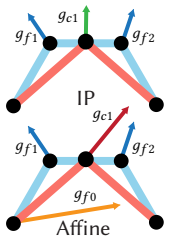
$$\gamma^i = \bar{B}^{-1}(\bar{x}_{fi} - \bar{x}_{c1}),\ (\beta_0^i, \beta_2^i)^T = \gamma^i \text{ and } \beta_1^i = 1 - (\beta_0^i + \beta_2^i).$$

Affine prolongation for a single collapse is then

$$\begin{bmatrix} \bar{x}_{f0} \\ \bar{x}_{f1} \\ \bar{x}_{f2} \\ \bar{x}_{f3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \beta_0^1 & \beta_1^1 & \beta_2^1 \\ \beta_0^2 & \beta_1^2 & \beta_2^2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{x}_{c0} \\ \bar{x}_{c1} \\ \bar{x}_{c2} \end{bmatrix},$$

and its corresponding restriction for downsampling arbitrary force fields g is

$$\begin{bmatrix} g_{c0} \\ g_{c1} \\ g_{c2} \end{bmatrix} = \begin{bmatrix} 1 & \beta_0^1 & \beta_0^2 & 0 \\ 0 & \beta_1^1 & \beta_1^2 & 0 \\ 0 & \beta_2^1 & \beta_2^2 & 1 \end{bmatrix} \begin{bmatrix} g_{f0} \\ g_{f1} \\ g_{f2} \\ g_{f3} \end{bmatrix}.$$



Here, we see that the force contributions to each coarse node, from the fine level nodes, are determined "volumetrically" from their spatial coordinates with respect to a coarse-mesh "triangle". The proximity of the shell's nodes in Euclidean space is inappropriately considered and produces spurious force contributions (see inset "Affine" example). In 3D, the situation largely parallels the 2D case with the least-squares affine map, providing a "volumetric" fit of a local frame for barycentric coordinates with respect to the coarse mesh, while fill-in for each stencil generally worsens. Irrespective of dimension, the influence of the fine mesh on coarse-level restricted forces broadens as nested edge collapses progress to form increasingly dense prolongation operators. As a result, coarse nodal forces (e.g., g_{c1}) are contributed to by an increasing number of fine-level forces. In turn, these finer-level forces are also net forces, again characterized, as covered above, by arbitrary directions and magnitudes, which can result in large restricted force magnitudes and noisy directions as the neighborhood of fine mesh nodes expands due to edge collapses.

In contrast, Intrinsic Prolongation (IP) computes its prolongation mapping by jointly flattening the two local patches onto the common domain to determine the *intrinsic* coordinates of the vertices on the fine mesh. This feature generates a *local* stencil for the coarse mesh's fine-level dependence. Concretely, (see inset "IP" example) the force acting on the middle coarse node g_{c1} is a weighted average of the fine nodal forces g_{f1} and g_{f2} based on its intrinsic coordinates. In contrast to the volumetric approach of affine fitting, only the fine nodes whose intrinsic counterparts exist in the shell's direct neighboring 2D lines (respectively 3D triangles) of the coarse node contribute to the restricted forces. Applying Intrinsic Prolongation (IP) for restriction thus applies a local and effective averaging for force downsampling.

B PSEUDOCODE

In this section, we present pseudocodes outlining the core PSQ methods. The remaining subroutines, while standard, are included here to ensure a comprehensive overview:

- minDistance$(x_d, \hat{C})$ - determine the smallest distance among all active primitive pairs.
- ConstructLookUpTable$(\mathcal{B})$ - build a specialized data structure for keeping track of the edges that need to be extended at each vertex based on the forward mesh decimation records $\mathcal{B}$.
- UpdateLookUpTable$(\mathcal{E})$ - update the pointer that indicates the next edges to be expanded at each vertex.

Algorithm 1 Optimization for Level-l Proxy Steps ▷ Section 4.5

procedure ProxyStep($x_l, y_l, n_l, \epsilon, \gamma, u$, max_iter)
  UpdateSystem(u) ▷ update materials, geometry and BCs
  $K_l(\cdot) \leftarrow K_l(\cdot, y_l)$ ▷ define the per-level inertial energy
  $i \leftarrow 0$
  while $i <$ max_iter **do**
    $\hat{C} \leftarrow$ ComputeConstraintSet(x_l)
    $H \leftarrow \frac{1}{h^2} M_l +$ ProjectPSD$\left(\nabla^2 E_l(x_l)\right)$
    $g \leftarrow (U^l)^T \nabla G(U^l x_l + a^l(x_l)) + \nabla C_l(x_l) + \nabla K_l(x_l)$
    $d \leftarrow -H^{-1} g$
    if $\|d\|_2 < \sqrt{n_l} h\epsilon$ **then** break
    end if
    $\alpha \leftarrow \min\left(1, \text{StepSizeFilter}(x_l, d, \hat{C})\right)$
    $\alpha \leftarrow$ LineSearch(F_l, x_l, d, α)
    if $\alpha < \gamma$ **then** break
    end if
    $x_l \leftarrow x_l + \alpha d$
    $i \leftarrow i + 1$
    if $i >$ max_iter **then** break
    end if
  end while
  eq $\leftarrow$ (i == 0) ▷ equilibrium at x_l if no iterations taken
  return $(x_l$, eq)
end procedure

Algorithm 2 Feasible Initialization ▷ Section 4.4

procedure FeasibleInit(x_l)
  $\mathcal{S} \leftarrow$ GraphColoring($\mathcal{M}_l$, $\mathcal{M}_{l+1}$, $\mathcal{B}$) ▷ precomputation
  $\mathcal{M}_d \leftarrow \mathcal{M}_l$, $x_d \leftarrow x_l$ ▷ dynamically changing
  for each color c in $\mathcal{S}$ **do**
    $\mathcal{M}_d \leftarrow$ UpdateTopology($\mathcal{M}_d, c$)
    $x_d \leftarrow$ SafeParallelExpansion(x_d, c)
  end for
  if ∃ strain limiting constraints **then**
    $x_{l+1} \leftarrow$ StrainLimitRelaxation(x_d) ▷ Section 4.4.5
  end if
end procedure

Algorithm 3 Safe Parallel Expansion ▷ Section 4.4.4

procedure SafeParallelExpansion(x_d, c)
  $\hat{C} \leftarrow$ ComputeConstraintSet(x_d)
  $g_0 \leftarrow$ minDistance($x_d, \hat{C}$)
  $\hat{x}_d \leftarrow$ ProlongedTargetPositions(x_d)
  for each new inserted vertex v with color c **do** ▷ parallel
    SafePerturb(v, $\hat{v}$, $0.01g_0$)
  end for
  do
    $d \leftarrow \hat{x}_d - x_d$
    $\alpha \leftarrow$ RatioStepSizeUpperBound($x_d, d, \hat{C}, g_0$)
    $x_d \leftarrow x_d + \alpha d$
    for vertices i in stencils that bound step size **do**
      $(\hat{x}_d)_i \leftarrow (x_d)_i$ ▷ bound reached
    end for
  while $\alpha < 1$
  $g_1 \leftarrow$ minDistance($x_d, \hat{C}$) ▷ no need for updating $\hat{C}$
  if $g_1 < \epsilon$ **then** ▷ ϵ denotes a minimum gap to enforce
    $x_d \leftarrow$ LocalRelaxationSteps(x_d)
  end if
end procedure

Algorithm 4 Ratio-Based Step Size Upper Bound ▷ Section 4.4.2

procedure RatioStepSizeUpperBound($x_d, d, \hat{C}, g$)
  for each primitive pair p in $\hat{C}$ **do**
    $d_0 \leftarrow$ distance(x_d, p)
    if $d_0 \ll g$ **then**
      $r \leftarrow 0.999$
    else
      $r \leftarrow 0.1$
    end if
    $d_1 \leftarrow$ AdditiveCCD(x_d, d, r) ▷ ensure $d_1 \geq rd_0$
  end for
end procedure

Algorithm 5 Stratified Random Sampling for Safe Perturbation ▷ Section 4.4.1

procedure SafePerturb(v, $\hat{v}$, ϵ)
  $v_n \leftarrow v + \epsilon(\hat{v} - v)/||\hat{v} - v||$ ▷ add perturbation
  while CheckLocalSelfIntersection(v_n) **do**
    $d \leftarrow$ GenerateRandomDirection() ▷ normalized
    $v_n \leftarrow v + \epsilon d$
  end while
  $v \leftarrow v_n$
end procedure

Algorithm 6 Dynamic Mesh (Graph) Coloring ▷ Section 4.4.4

procedure GraphColoring($\mathcal{M}_l$, $\mathcal{M}_{l+1}$, $\mathcal{B}$)
  $\mathcal{M}_d \leftarrow \mathcal{M}_l$ ▷ dynamically changing
  $\mathcal{E} \leftarrow$ ConstructLookUpTable($\mathcal{B}$)
  $\mathcal{S} \leftarrow \{\}$
  while $\mathcal{M}_d \neq \mathcal{M}_{l+1}$ **do**
    $\boldsymbol{s} \leftarrow$ IndependentSet($\mathcal{M}_d, \mathcal{B}$)
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{s}\}$
    $\mathcal{M}_d \leftarrow$ ExpandEdges($\mathcal{M}_d, \mathcal{E}, \boldsymbol{s}$)
    UpdateLookUpTable($\mathcal{E}$)
  end while
  return $\mathcal{S}$
end procedure

Algorithm 7 Construction of an Independent Set ▷ Section 4.4.4

procedure IndependentSet($\mathcal{M}_d, \mathcal{B}$)
  $\boldsymbol{s} \leftarrow \{\}$ ▷ collect the independent vertices
  $\boldsymbol{n} \leftarrow \{\}$ ▷ collect the neighbors of vertices in $\boldsymbol{s}$
  for each vertex v in $\mathcal{M}_d$ **do**
    $\mathcal{R} \leftarrow$ LocalOneRing($v, \mathcal{B}$)
    if $\forall r \in \mathcal{R}$ $r \notin \boldsymbol{n}$ **and** r in $\mathcal{M}_d$ **then**
      $\boldsymbol{s} \leftarrow \boldsymbol{s} \cup v$
      $\boldsymbol{n} \leftarrow \boldsymbol{n} \cup \mathcal{R}$
    end if
  end for
  return $\boldsymbol{s}$
end procedure