Progressive Shell Quasistatics for Unstructured Meshes

JIAYI ERISS 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...
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.


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.

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 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 fractionally 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; Gribin et al. 2003; Harmon et al. 2009; Li et al. 2020b; Narain et al. 2012; Tzaronopolous 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 et al. 2020; Li et al. 2020b; 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 et al. 2020; Ly et al. 2020; Zhang et al. 2019], and high-performance parallel implementations [Li et al. 2020b; Schmidt 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; Xiont et al. 2019]. Such multigrid solvers and preconditioners are complementary to Progressive Shell Quasistatics, 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 piecwise 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 formulate 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 Kobelt 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 Kobelt 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 simple 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 Kobelt [2003] — separate coarse surface changes from...
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; Barbić 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.
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\footnote{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.} with shell ($\Psi$), 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, $E_l(x) = E_j(x, \bar{x}, u)$ 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,

$$x^{t+1} = \arg\min_{x} \frac{1}{2h^2} \| x - x^t \|^2_M + E(x, \bar{x}, u^{t+1}),$$

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

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

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

$$F_j(x_l) = \underbrace{B_j(x_l)}_{C_l(x_l)} + \underbrace{D_j(x_l)}_{C_l(x_l)} + \underbrace{S_j(x_l)}_{C_l(x_l)} + \underbrace{\Psi_j(P^l_{l+1}(x_l))}_{C_l(x_l)}.$$

Here shell elastics, $\Psi_l$, are evaluated at the finest-resolution model, via a direct prolongation, $P^l_{l+1}(x_l)$, 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

$$x^{t+1}_l = \arg\min_{x_l} \frac{1}{2h^2} \| x_l - x^t_l \|^2_M + F_j(x_l).$$

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, $x^{t+1}_{l+1} = P^l_{l+1}(x^t_l)$, after which we proceed by quasistatic stepping with solves of (3). However, while this prolonged geometry provides a natural starting point...
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 \( x_i^{t+1} \) in order to begin the new level’s preview stepping (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, \( p_l \in \mathbb{R}^{3n_l \times 3n_l}, \) 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, \( p_L(x_l) = p_{L-1} \cdots p_{L+1}x_l. \)

For safe initialization PCS’s nested hierarchy also provides a safe and simple, intersection-free initializer, \( x_i^{t+1} \), via Barycentric upsample followed by linear search along \( d = p_l(x_l) - x_i^{t+1} \) 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.
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 similar 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 (uo) coordinates that will anchor the locations of the two newly split vertices (for the next l+1 level mesh), ˘o<sub>1</sub> and ˘o<sub>2</sub>, in the current mesh l.

We then store, one-time, the rest-shape extrinsic difference, ˘h<sub>i</sub> = s<sub>i+</sub> - ˘o<sub>i</sub>, per new-level vertex i, with respect to the coarse level l’s rest-shape triangle containing ˘o<sub>i</sub> (Figure 8). With ordered vertex rest-positions, ˘x<sub>0</sub>, ˘x<sub>1</sub>, ˘x<sub>2</sub>, for the containing triangle, we decompose the extrinsic difference into an out-of-plane offset ˘y<sub>i</sub> = n<sub>i</sub>˘h<sub>i</sub> ∈ R along the triangle’s normal, n, and the remaining, in-plane contribution ˘t<sub>i</sub> ∈ R<sup>2</sup>, so that ˘h<sub>i</sub> = ˘y<sub>i</sub> + (˘n × ˘e)˘t<sub>i</sub>, with ˘e = ˘x<sub>i</sub> − ˘x<sub>0</sub>∥∥˘x<sub>i</sub> − ˘x<sub>0</sub>∥.

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, s<sub>0+</sub>. Bending of our shell’s hinge elements then determines each containing triangle’s new normal direction, n<sub>i</sub>(˘x<sub>i</sub>), for application of our out-of-plane contribution, ˘y<sub>i</sub>. Finally, and correspondingly, the rotational factor, R<sub>i</sub>(˘F) ∈ R<sup>3x3</sup>, of each containing triangle’s deformation gradient, ˘F, gives the rotation for our remaining, in-plane offset, ˘t<sub>i</sub>. See the inset for an illustration of this process.

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

\[
x<sub>i+</sub> = ˘g<sub>i</sub> + a<sub>i</sub>(˘x<sub>i</sub>),
\]

where the nonlinear offset is

\[
a<sub>i</sub>(˘x<sub>i</sub>) = n<sub>i</sub>(˘x<sub>i</sub>)˘y<sub>i</sub> + R<sub>i</sub>(˘x<sub>i</sub>)˘t<sub>i</sub>,
\]

and (˘y<sub>i</sub>, ˘t<sub>i</sub>) ∈ R<sup>3</sup> 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, ˘x<sub>i+</sub> = U<sub>i+</sub>˘x<sub>i</sub>. We then can similarly apply our nonlinear offset computation globally for all m<sub>i</sub> expanded nodes as

\[
˘d<sub>i</sub> = [a<sub>1</sub>(˘x<sub>1</sub>)T, \ldots, a<sub>m<sub>i</sub></sub>(˘x<sub>m<sub>i</sub></sub>)T]T,
\]

so that our per-level shell-prolongation operator is

\[
x<sub>i+</sub> = U<sub>i+</sub>˘x<sub>i</sub> + ˘d<sub>i</sub> = U<sub>i+</sub>˘x<sub>i</sub> + a<sub>i</sub>(˘x<sub>i</sub>).
\]
As discussed in §3, the Progressive Simulation framework requires (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 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, \( P_{l+1}^L(x_i^{l+1}) \). 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 \( P_{l+1}^L(x_i^{l+1}) \). 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.).
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, \( \chi_i^{k+1} \), 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 \( 10^{-3} \)) by keeping the IPC contact barrier stiffness, \( k \), 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.

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 its 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 $M_1$ to mesh $M_{l+1}$ 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, require the solution of a challenging, global detangling problem with expensive global intersection queries and sampling.

Instead, we decompose the expansion process from $M_1$ to $M_{l+1}$ 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, $\delta_i$. 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, $\alpha$. 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 $\alpha$. 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, $\alpha$. 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 $\alpha$. We do this by canceling their remaining displacement, $\delta_j \leftarrow 0$. 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 our vertex 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, $P^l_{l+1}(x_i^{l+1})$. 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, $M_l$, and iteratively optimize our newly inserted vertices from $M_{l+1}$ 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, $H(x_l) = \frac{1}{\ell} \ell M_l + \text{ProjectPSD}(\nabla^2 E(x_l))$, 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, $\delta_j = \delta^a(x_l)$. Each iteration’s search direction, $p$, is then obtained by...
We implement our methods in C++, applying PARDISO [Bollhöfer et al. 2010] while both the PCS and PSQ preview solutions well-capture their targets during each solve iteration. Collision objects are then treated identically to all the other input meshed domains. Then, at each level of simulation, we simply record the per-step target positions \( \mathbf{x} \) and \( \mathbf{d} \) as 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 preprocessed 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 progresses-simulation pipeline.

5 EVALUATION

We implement our methods in C++, applying PARDISO [Bollhöfer et al. 2010], 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

solving the linear system

\[
H(x) \Delta p = -\left( \frac{1}{h^2} \mathbf{M} + (U^i)^T \nabla \Psi(U^i x + d) + \nabla C(x) \right).
\]

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, \( ||\Delta p|| < \epsilon \) where \( \alpha \) 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.

As covered in §4, intrinsic 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
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.

**Plate Tests:** Bending and shearing “unit tests” with a stiff metal sheet.

**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;
Progressive Shell Quasistatics for Unstructured Meshes

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 upscaled 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 reinforce-ments. 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 Huesh-Ti Derek Liu for insightful discussion and help in making Figure 18.

REFERENCES


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).


3. Increase the number of fans


A 2D RESTRICTION ANALYSIS

1. Make a star
2. Twist until nearly intersecting
3. Increase the number of fans
4. Compress

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’s) significant improvement over affine prolongation for downsampling forces. Consider the simple 2D example in Figure 21 with a fine polyl ine 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 (x0, x1, x2) and deformed (x0, x1, x2). Similarly, rest and deformed fine vertices are (xf0, xf1, xf2) and (xf0, xf1, xf2).
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 PSEUOCODE

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($x_d, \tilde{C}$) - determine the smallest distance among all active primitive pairs.
- **ConstructLookUpTable($B$) - 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 $B$.
- **UpdateLookUpTable($E$) - update the pointer that indicates the next edges to be expanded at each vertex.

Algorithm 1 Optimization for Level-I Proxy Steps

1. **procedure** ProxyStep($x_l, y_l, n_l, \epsilon, v, u, \text{max}_\text{iter}$)
2. **UPDATE SYSTEM**($u$) - update materials, geometry and BCs
3. $K_l(\cdot) \leftarrow K_l(\cdot, y_l)$ - define the per-level inertial energy
4. $i \leftarrow 0$
5. **while** $i < \text{max}_\text{iter}$ **do**
6. $\tilde{C} \leftarrow \text{COMPUTE CONSTRAINT SET}(x_l)$
7. $H \leftarrow \frac{1}{M_l} M_l + \text{PROJ} \text{ECT PSD} (\nabla^2 E_l(x_l))$
8. $g \leftarrow (U_l^T \nabla G(U_l x_l + a l(x_l))) + \nabla C_l(x_l) + \nabla K_l(x_l)$
9. $d \leftarrow -H^{-1} g$
10. **if** $\|d\|_2 < \sqrt{\mu} h$ **then** **break**
11. **end if**
12. $\alpha \leftarrow \min \left(1, \text{STEP SIZE FILTER}(x_l, d, \tilde{C}) \right)$
13. $\alpha \leftarrow \text{LINE SEARCH}(F_l, x_l, d, \alpha)$
14. **if** $\alpha < \gamma$ **then** **break**
15. **end if**
16. $x_l \leftarrow x_l + \alpha d$
17. $i \leftarrow i + 1$
18. **if** $i > \text{max}_\text{iter}$ **then** **break**
19. **end if**
20. **end while**
21. $\epsilon \leftarrow \text{STEP SIZE FILTER}(x_l, \tilde{C})$
22. **return** $(x_l, \epsilon)$
23. **end procedure**
Algorithm 2 Feasible Initialization → Section 4.4
1: procedure FeasibleInit(x_i)
2: \( S \leftarrow \text{GraphColoring}(M_i, M_{i+1}, B) \) \( \triangleright \) precomputation
3: \( M_d \leftarrow M_i, x_d \leftarrow x_i \) \( \triangleright \) dynamically changing
4: for each color \( c \) in \( S \) do
5: \( M_d \leftarrow \text{UpdateTopology}(M_d, c) \)
6: \( x_d \leftarrow \text{SafeParallelExpansion}(x_d, c) \)
7: end for
8: if \( \exists \) strain limiting constraints then
9: \( x_{i+1} \leftarrow \text{StrainLimitRelaxation}(x_d) \) \( \triangleright \) Section 4.4.5
10: end if
11: end procedure

Algorithm 3 Safe Parallel Expansion → Section 4.4.4
1: procedure SafeParallelExpansion(x_d, c)
2: \( g_i \leftarrow \text{ComputeConstraintSet}(x_d) \)
3: \( g_i \leftarrow \minDistance(x_d, \hat{C}) \)
4: \( x_d \leftarrow \text{ProlongedTargetPositions}(x_d) \)
5: for each new inserted vertex \( v \) with color \( c \) do \( \triangleright \) parallel
6: \( \text{SafePerturb}(c, \delta, \epsilon, g_i) \)
7: end for
8: do
9: \( d \leftarrow \hat{x}_d - x_d \)
10: \( a \leftarrow \text{RatioStepSizeUpperBound}(x_d, d, \hat{C}, g_i) \)
11: \( x_d \leftarrow x_d + ad \)
12: for vertices \( i \) in stencils that bound step size do
13: \( (\hat{x}_d)_i \leftarrow (x_d)_i \) \( \triangleright \) bound reached
14: end for
15: while \( a < 1 \)
16: \( g_i \leftarrow \minDistance(x_d, \hat{C}) \) \( \triangleright \) no need for updating \( \hat{C} \)
17: if \( g_i < \epsilon \) then \( \triangleright \) \( \epsilon \) denotes a minimum gap to enforce
18: \( x_d \leftarrow \text{LocalRelaxationSteps}(x_d) \)
19: end if
20: end procedure

Algorithm 4 Ratio-Based Step Size Upper Bound → Section 4.4.2
1: procedure RatioStepSizeUpperBound(x_d, \( d, \hat{C}, g \))
2: for each primitive pair \( p \) in \( \hat{C} \) do
3: \( d_0 \leftarrow \text{Distance}(x_d, p) \)
4: if \( d_0 < g \) then
5: \( r \leftarrow 0.999 \)
6: else
7: \( r \leftarrow 0.1 \)
8: end if
9: \( d_1 \leftarrow \text{AdditiveCCD}(x_d, d, r) \) \( \triangleright \) ensure \( d_1 \geq rd_0 \)
10: end for
11: end procedure

Algorithm 5 Stratified Random Sampling for Safe Perturbation → Section 4.4.4.1
1: procedure SafePerturb(\( \hat{v}, \delta, \epsilon, \alpha \))
2: \( v_n \leftarrow \hat{v} + \alpha (\hat{v} - v)/\|\hat{v} - v\| \) \( \triangleright \) add perturbation
3: while \( \text{CheckLocalSelfIntersection}(v_n) \) do
4: \( d \leftarrow \text{GenerateRandomDirection}() \) \( \triangleright \) normalized
5: \( v_n \leftarrow v + ed \)
6: end while
7: \( v \leftarrow v_n \)
8: end procedure

Algorithm 6 Dynamic Mesh (Graph) Coloring → Section 4.4.4
1: procedure GraphColoring(M_i, M_{i+1}, B)
2: \( M_d \leftarrow M_i \) \( \triangleright \) dynamically changing
3: \( E \leftarrow \text{ConstructLookUpTable}(B) \)
4: \( S \leftarrow \{ \} \)
5: while \( M_d \neq M_{i+1} \) do
6: \( s \leftarrow \text{IndependentSet}(M_d, B) \)
7: \( S \leftarrow S \cup \{ s \} \)
8: \( M_d \leftarrow \text{ExpandEdges}(M_d, E, s) \)
9: \( \text{UpdateLookUpTable}(E) \)
10: end while
11: return \( S \)
12: end procedure

Algorithm 7 Construction of an Independent Set → Section 4.4.4
1: procedure IndependentSet(M_d, B)
2: \( s \leftarrow \{ \} \) \( \triangleright \) collect the independent vertices
3: \( n \leftarrow \{ \} \) \( \triangleright \) collect the neighbors of vertices in \( s \)
4: for each vertex \( v \) in \( M_d \) do
5: \( R \leftarrow \text{LocalOneRing}(v, B) \)
6: if \( \forall r \in R \ r \notin n \) and \( r \in M_d \) then
7: \( s \leftarrow s \cup v \)
8: \( n \leftarrow n \cup R \)
9: end if
10: end for
11: return \( s \)
12: end procedure