Nested Cages

Leonardo Sacht[1] Etienne Vouga[2] Alec Jacobson[3]

[1]Universidade Federal de Santa Catarina [2]University of Texas at Austin [3]Columbia University

Figure 1: *Given an input shape (yellow on bottom right), our method constructs* nested *cages: each subsequent mesh is coarser than the last and fully encloses it without intersections. A slice through all layers (left), shows a tightly encaged* Bunny.

Abstract

Many tasks in geometry processing and physical simulation benefit from multiresolution hierarchies. One important characteristic across a variety of applications is that coarser layers strictly *encage* finer layers, nesting one another. Existing techniques such as surface mesh decimation, voxelization, or contouring distance level sets do not provide sufficient control over the *quality* of the output surfaces while maintaining strict nesting. We propose a solution that enables use of application-specific decimation and quality metrics. The method constructs each next-coarsest level of the hierarchy, using a sequence of decimation, flow, and contact-aware optimization steps. From coarse to fine, each layer then fully encages the next while retaining a snug fit. The method is applicable to a wide variety of shapes of complex geometry and topology. We demonstrate the effectiveness of our nested cages not only for multigrid solvers, but also for conservative collision detection, domain discretization for elastic simulation, and cage-based geometric modeling.

CR Category: I.3.0 [Computer Graphics]

Keywords: Mesh decimation, geometric flow, multigrid

1 Introduction

As the complexity and size of computational objects continue to grow, acceleration algorithms become increasingly important. One powerful technique is to decompose a high-resolution mesh into a hierarchy of increasingly coarse approximations or *cages* (see Figure 1). For example, multigrid FEM techniques efficiently solve Eulerian PDEs on very fine meshes by moving up and down the hierarchy; low frequency residual error disappears quickly on the coarsest levels while fine levels smooth away high frequency error. Coarse enclosing cages also find use in physical simulation, where deformations of the cage are interpolated onto embedded high-resolution geometries; in interactive animation, where artists specify large-scale deformations by adjusting the low-dimensional cage; and in collision detection, where conservative culling reduces computation time. For all these applications, the key to high performance is the ability to generate a quality multiresolution hierarchy.

The straightforward approach to building a hierarchy around an object F is to use an application-specific decimation algorithm to build a coarse approximation C to F; C itself can be further coarsened to build the next level of the hierarchy, etc. (see Figure 2). Unfortunately C will typically intersect F, which is often undesirable: most algorithms for transferring the pose of a deformation cage to a detailed object only guarantee small distortion of the object and lack of element-inversion artifacts if the object is entirely contained within the cage [Joshi et al. 2007; Ben-Chen et al. 2009b]; strict nesting is essential for accelerating collision detection *conservatively*; and while nesting is not a necessary condition for multigrid convergence [Chan et al. 1996], the ability to use simple linear interpolation for prolongation is known to be more robust, more efficient, and easier to implement [Chan & Wan 2000; Dickopf 2010]. This is particularly important for enforcing Neumann boundary conditions, common to simulation and geometry processing [Chan et al. 1999].

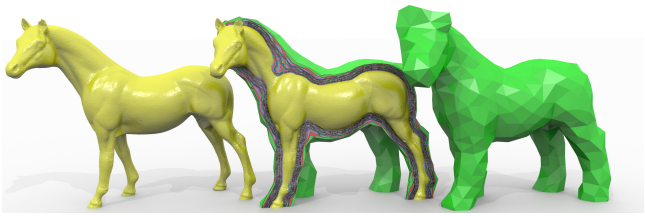


Figure 2: *25 increasingly coarse cages nested around a* Horse. *Our tight nesting property is robust even for a large number of cages.*

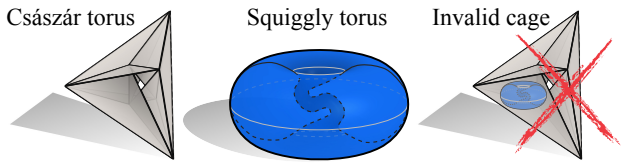
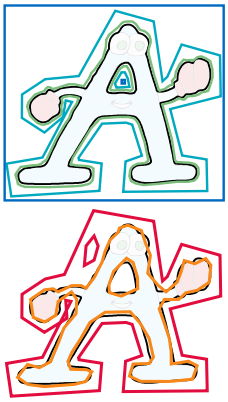


Figure 3: *Some coarse-fine mesh combinations are impossible. The* Császár torus *is too coarse to nested around the* Squiggly torus*'s complex handle. While embedding the* Squiggly torus inside *the* Császár torus *is possible (right), this is not a validly* nesting *cage.*

For these reasons, one would like a cage C that is *nested* around F: that surrounds F without intersecting it. More formally, given two watertight (connected, closed, self-intersection-free and oriented) meshes C and F (later we generalize our method to build cages for polygon soups), we say that C *nests* F if: (i) F is contained in the interior of C, and (ii) for every loop on C, there is a homotopy from that loop to a loop on F that remains at all times inside C's enclosed volume. In other words, for every handle of C there is a corresponding one on F, but not necessarily the other way around. This definition extends naturally to when F is a polyhedron with voids or multiple connected components.



The inset figure shows examples of cages that do (top) and do not (bottom) nest a 2D example (black outline). Given a $\widehat{C}$ that does not nest F, we look for the best perturbation C of $\widehat{C}$ that does nest F; where "best" is measured by an application-dependent fitness energy $E(F, \widehat{C}, C)$, penalizing, for example, total volume for tight fitting cages for efficient multigrid; distortion for simulation or collisions; or violations of symmetry and quad-planarity for interactive deformation cages. Finding C entails solving the *nested cage problem*:

Given an embedded polyhedron F, a polyhedron $\widehat{C}$, and a fitness function E, the goal is to minimize $E(F, \widehat{C}, C)$ over all embeddings C of $\widehat{C}$ that nest F.

This problem in its full generality is intractable. For single polygons in 2D, and simply-connected meshes in 3D, at least one nested embedding C always exists, but once F is allowed to have nontrivial topology, counterexamples exist where it is impossible to embed $\widehat{C}$ in a way that nests F (see Figure 3). The decision problem of whether any nesting of $\widehat{C}$ around F exists can be shown to be NP-complete [Vouga et al. 2015]. Fortunately, in practice $\widehat{C}$ and F are not completely arbitrary; any reasonable decimation scheme will keep $\widehat{C}$ morphologically similar, and roughly aligned, to F, and these favorable initial conditions can be leveraged by heuristics.

Contribution. We present a practical algorithm for solving the nested cage problem on meshes typically used in applications such as physical simulation and geometric modeling. Our method remains completely *agnostic* to the decimation scheme used to create $\widehat{C}$ and fitness function E. Briefly, the method first flows F inside $\widehat{C}$ along $\widehat{C}$'s signed distance field; once inside, we rewind the flow while deforming $\widehat{C}$ to minimize E; contact forces guarantee that the coarse cage always encloses F at each reverse-flow time step. This process can be generalized to building an entire multiresolution hierarchy around F. Although our method is not guaranteed to always find a solution C, especially for very coarse or nearly-self-intersecting F (see Section 5 for discussion of failure cases), we tested our algorithm on an extensive zoo of example meshes (see Section 4).

2 Related methods and special cases

Our method solves the nested cage problem for the large variety of input meshes shown in Section 4 *while* interoperating with arbitrary problem-specific decimation algorithms and fitness functions. Many related approaches have been proposed that work for special cases, or relax requirements imposed above.

If the algorithm is free to generate the coarse cage $\widehat{C}$, rather than accepting it as input from the user, the problem becomes substantially easier, and has been extensively studied:

Bounding polytopes. Most naively, one can simply take $\widehat{C}$ to be a canonical bounding polytope (e.g. box, KDOP, convex hull). An extension would be to "shrink-wrap" the bounding polytope around $\widehat{F}$ to minimize E. This idea has been explored before [Peterhans 2012; Wang et al. 2013], but no full solution has been developed. This is presumably because—while this approach works well for convex $\widehat{F}$—it will not give a good fit for meshes with nooks or concavities. It is also not clear how to find a bounding polytope for meshes of nontrivial topology. Another approach might be to stitch overlapping convex volumes [Xian et al. 2012] using mesh boolean operations, but cage topology and *quality* become difficult to control without sacrificing control of resolution.

Offset surfaces for simulation meshes. Representing $\widehat{F}$ as an implicit surface and *contouring level sets* is a popular method of creating offset surfaces that nest $\widehat{F}$, particularly in the context of simulations that use a "simulation mesh" as a proxy for very fine rendered geometry [Campen & Kobbelt 2010]. The challenge is not just defining a scalar function with an isosurface enveloping the object, but also contouring it with a piecewise-linear triangle mesh without (i) cutting corners too harshly and intersecting $\widehat{F}$ or (ii) resorting to fine resolution, nullifying any performance gains of using the cage. Xu et al [2014] define a robust signed distance field, but pass the field to an off-the-shelf contour mesher without any attempt to guarantee non-intersection with the input. Similarly, Shen at al [2004] iteratively refine a moving least squares iso-surface to enclose an existing model tightly via a global scaling parameter.

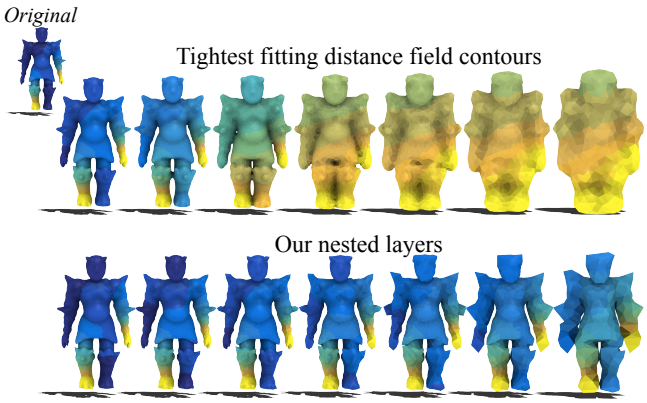


Figure 4: *Contouring requires aggressive spacing between distance-field isolevels to produce valid nesting. Semantically distant parts* fuse *together, destroying shape-awareness, visualized by a pseudo-coloring of a Poisson solution computed on each level.*

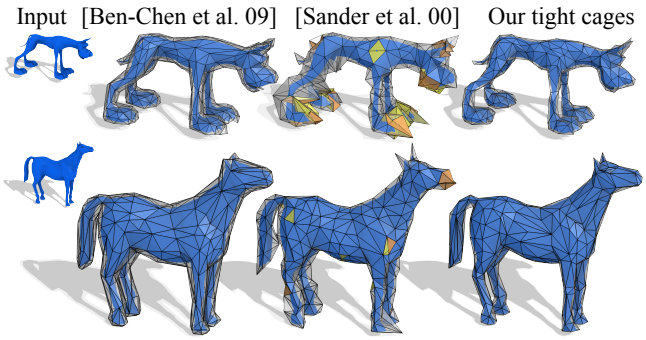


Figure 5: *Methods like [Ben-Chen et al. 2009a] use coarse cages as computational workhorses to reduce complexity. Their iterative offset heuristic does not allow control of the cage topology. Meanwhile, [Sander et al. 2000] attempt to maintain nestedness during greedy decimation with local constraints. Cages are loosely-fitting, but more importantly self-intersections (orange) and intersections with the input model (yellow) eventually accumulate.*

Some applications, such as collision detection, can make use of implicit representations without intermediary meshes, but when a mesh is needed it is not enough that the *continuous* isosurface does not intersect the input model since generic contouring will invalidate this. A large enough iso-value or small-enough resolution tolerance must always be chosen to ensure non-intersection after contouring. For large iso-values, topological control is lost and close features are quickly merged (see Figure 4), violating the nestedness of the cage. A similar approach [Ben-Chen et al. 2009a] has been used for building deformation cages around input shapes, where an offset surface is created via Poisson surface reconstruction [Kazhdan et al. 2006] (we compare against this approach in Figure 5). While the tightness of these cages could be controlled with post-hoc shrinking of the cage, the method can introduce topology changes that are not easily remedied. For meshes with well defined feature curves, it may be possible to merge coarse on-surface triangulations of patches [Xian et al. 2013], though strict nesting is not sought or guaranteed.

Progressive decimation of $\widehat{F}$ using edge collapses, taking care to place new vertices on the exterior of the current volume by solving a system of inequality constraints [Sander et al. 2000], has found some success in real-time rendering and collision detection [Platis & Theoharis 2003]. Although the *vertices* of these "progressive hulls" are guaranteed to lie outside of $\widehat{F}$, edges and faces of $\widehat{C}$ might still intersect $\widehat{F}$ (and $\widehat{C}$ might globally self-intersect); see Figure 5. We tested this method on the entire "zoo" of examples shown in Figure 11 and the supplementary material, and of the 26 examples there, in only one case was the entire hierarchy free of such intersections. We also observe that for coarse cages these hulls tend to be more loose-fitting than cages produced by methods that optimize the cage shape globally (see Figure 5). Self-intersections resulting from edge collapses can be corrected with post-hoc mesh repair [Deng et al. 2011], but this technique relies on temperamental 3D tetrahedral meshing in tight regions near overlaps and does not consider face- or edge-intersections with the input model.

Voxelization of $\widehat{F}$ will create a nested cage, provided that the resolution is chosen fine enough to avoid topological artifacts (as in the case of contouring implicit functions). Naive voxelization yields dense, inefficient cages [Mehra et al. 2009], possibly improved by progressive decimation [Xian et al. 2009] or mesh booleans [Xian et al. 2015] though also inheriting their respective drawbacks.

Mesh untangling. All of the above methods require relinquishing some degree of control over decimation to the nested cage algorithm;

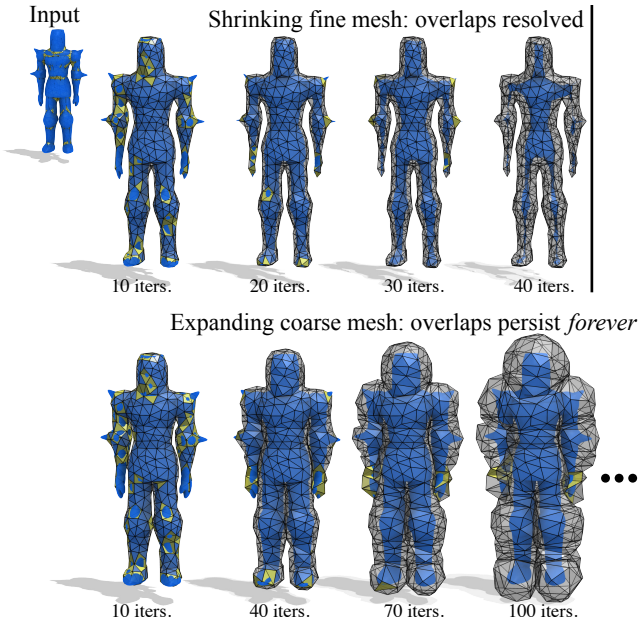


Figure 6: *Shrinking the fine mesh inside the coarse quickly removes overlaps (yellow). Expanding the coarse mesh requires self-contact handling (e.g. between arms and body), blocking progress.*

algorithms that accept an arbitrary pre-decimated cage $\widehat{C}$ are less well-developed. *History-free* cloth collision response methods (e.g. [Baraff et al. 2003; Volino & Magnenat-Thalmann 2006; Wicker et al. 2006; Ye & Zhao 2012]) could be used to separate $\widehat{C}$ from $\widehat{F}$ if intersections are not too severe; otherwise such methods quickly get stuck in local minima.

Outward Flow. Before settling on an inward flow of $\widehat{F}$, we experimented with an outward flow of $\widehat{C}$ away from $\widehat{F}$, along its signed distance field. For convex meshes this works well. However, extensive testing revealed that this approach suffers from several difficulties on more complex geometries: 1) Flowing vertices along the signed distance field of $\widehat{F}$ is not guaranteed to resolve all face-face collisions between the fine and coarse meshes; the flow is most robust when the flowing elements are small and the signed distance field does not have too many fine features. Flowing the coarse mesh outward is more fragile than flowing the fine mesh in as both of these factors are less favorable; 2) when flowing the fine mesh inside the coarse mesh we do not need to handle self-collisions within the fine mesh, whereas if inflating the coarse mesh, we do. Adding collision constraints aggravates the fragility of the outward flow; the constraints can prevent a flowing face from ever leaving the volume of the fine mesh (see Figure 6).

Multigrid. Methods based on regular grids or lattices naturally support multiresolution numerical methods. McAdams et al. [2011] successfully employ multigrid on voxelized shapes to animate volumetric elastic characters. The main downside of grid-based methods is their traditionally poor handling of irregular boundaries. Specialized multigrid methods using adaptive octrees to handle complex boundaries for fractures [Dick et al. 2011] and fluids [Ferstl et al. 2014] exist, but are not immune to troubles of regular grids: boundaries must be represented at fine grid levels to avoid aliasing and retain the input's topology. To avoid *fusing* together geodesically distant parts, the shape must be modeled in an accommodating pose. If remodeling is not possible, the grid size must be chosen smaller than not just the smallest features but also the *smallest void* between features. One solution is to replicate cells in close areas and man-

age adjacencies when two or more replicated patches merge and split [Teran et al. 2005; Nesme et al. 2009; Sýkora et al. 2009]. Cell replication is not only difficult to realize robustly, but also riddles multigrid numerical methods with expensive, SIMD-breaking boundary handling code, detracting from the performance gains of memory-efficient regular grids [Demmel 2004].

Multigrid on unstructured grids or tetrahedral meshes is more temperamental, and constructing each level requires care [Fish et al. 1995]. Geometric multigrid schemes typically coarsen an input tetrahedral mesh by removing vertices, attempting to connect those remaining in a reasonable way [Guillard 1993; Adams & Demmel 1999]. Special care is required to maintain any semblance of the original boundary [Brune et al. 2011], essentially devolving into constrained Delaunay tessellation with no guarantee that the coarsening will not *eat away* large portions of the domain. In many scenarios, the boundary of the domain is assumed to be only as irregular as the coarsest layer, simplifying level design [Feng et al. 1997].

A second group of methods generates a multiresolution hierarchy using decimation. Unmodified mesh decimations [Garland & Heckbert 1997] have been used for adaptive simulations for visco-elastic solids [Debunne et al. 2001]. Such non-nested cages require extrapolation or one-to-many mappings to *prolongate* solutions on the coarse levels to finer levels. We show cases where this fails to converge for common linear systems on irregular domains, and we show better convergence for strictly nesting cages where prolongation is a purely linear interpolation.

Algebraic alternatives also exist [Ruge & Stüben 1987], but are recommended only when geometric information is not available [Falgout 2006]. Recently, [Krishnan et al. 2013] proposed a multigrid preconditioner for Laplace-based systems on images and surfaces. This method combines the elegance of algebraic techniques with some geometric information derived from the characteristics of the Laplacian matrix, but is limited to special problems.

Simplification. The majority of surface mesh decimation techniques aim to preserve outward appearance with lower and lower mesh resolution [Hoppe 1996; Garland & Heckbert 1997; Melax 1998]. Along these lines, [Gumhold et al. 2003] output decimations free of *self*-intersections, ensuring for example that a character's clothing stays outside its body. In contrast, our method assumes self-intersection free input decimations and transforms them into nested layers, with no intersections *across* hierarchy layers. Recent work has considered more elaborate decimation goals than appearance such as preserving haptic sensations [Otaduy & Lin 2003b]. Otdauy and Lin [2003a] also demonstrate how to combine mesh decimations and bounding volume hierarchies to achieve faster collision detection without affecting visual appearance. Our nested cages are hierarchical decimations strictly containing the input shape, ensuring strictly conservative collision detection. Rather than work against the sophistication of existing shape decimation techniques, we complement them. Our method takes arbitrary decimations as input and nests them as a post process.

Interference-aware processing. We credit [Harmon et al. 2011] for their ground-breaking introduction of contact handling to mainstream geometry processing. Their work inspires us to consider the contact and collisions tool-set familiar to physically based simulation in our geometry processing task. In this task, our novel flow is essential for finding a feasible starting state.

3 Method

The input to our method is a sequence of $k+1$ potentially overlapping triangle meshes $\widehat{M}_0, \widehat{M}_1, \ldots, \widehat{M}_k$. Each mesh $\widehat{M}_i$ has a corresponding list of *initial* vertex positions $\widehat{\mathbf{M}}_i \in \mathbb{R}^{n_i \times 3}$ and list of triangle indices $\mathbf{T}_i \in \{1, \ldots, n_i\}^{m_i \times 3}$. In a typical scenario, $\widehat{M}_0$ is a high-resolution *original* mesh and $\widehat{M}_1, \ldots, \widehat{M}_k$ are decimations of decreasing resolution; the method of decimation is unimportant so long as the input meshes all approximate $\widehat{M}_0$ and are all watertight. Depending on the application, troublesome input meshes can be *cleaned* as a pre-process using available tools (e.g. [Attene 2010; Jacobson et al. 2013]).

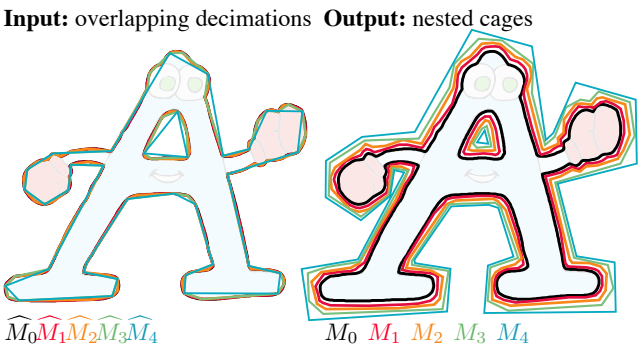


Figure 7: *We deform a set of input overlapping decimations so that each contains all finer layers (2D illustration).*

The output of our method is a new sequence of k lists of vertex positions $\mathbf{M}_1, \ldots, \mathbf{M}_k$ such that for $i = 1 \ldots k$ each $M_i = (\mathbf{M}_i, \mathbf{T}_i)$ is a deformed mesh whose surface nests M_{i-1} (with $M_0 = \widehat{M}_0$.) Recursively this ensures that each M_i lies strictly outside M_j for $j = i-1, \ldots, 0$. We call such meshes *nested* (see Figure 7).

We preserve the original *combinatorics* of $\widehat{M}_0, \widehat{M}_1, \ldots, \widehat{M}_k$ ($\mathbf{T}_i$ are unchanged), respecting the output of whichever problem-specific decimation routine produced them. The *nesting* property of the output meshes is easily verified by testing that at least one vertex (per connected component in the general case) of $\mathbf{M}_{i-1}$ lies inside M_i (e.g. has positive winding number) and that no intersections exist between M_{i-1} and M_i.

We now describe a general method that produces this nesting property while also optimizing any problem-specific energy E (see Algorithm 1, with additional subroutines in Appendix B).

Algorithm 1: nested_cages($M_0, L,$ Dec, Energy) $\rightarrow M_1, \ldots, M_k$

Inputs:

M_0	Initial high-res mesh, vertices $\mathbf{M}_0$, faces $\mathbf{T}_0$
L	k-long list of desired mesh resolutions
Dec	Function object for black-box decimator
Energy	Function object for re-inflation energy and gradient

Outputs:

M_1,...,M_k	k-long list of nested cages

begin

  for $i \in \{1, \ldots, k\}$ **do**

    (Decimate from previous layer or input mesh *)*

    $\widehat{M}_i \leftarrow \text{Dec}\left(M_{i-1}, \widehat{M}_{i-1}, L(i)\right)$

    $F \leftarrow M_{i-1}, \widehat{C} \leftarrow \widehat{M}_i$ *// Rename coarse and fine meshes*

    $H \leftarrow \text{Shrink}(\widehat{C}, F)$ *// history of shrinking fine mesh*

    (Reverse history, then M_i will* nest *M_{i-1} *)*

    $M_i \leftarrow \text{ReInflate}(H, \widehat{C}, \text{Energy})$

Our method operates recursively on two meshes of the sequence at a time: we compute M_i by considering only its original embedding

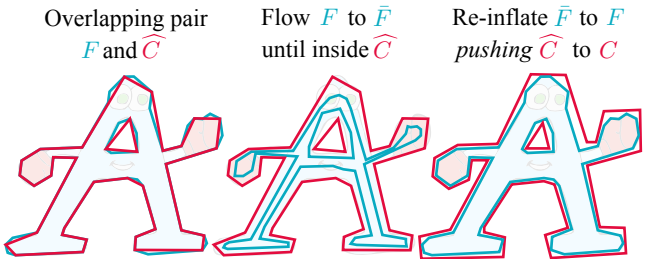


Figure 8: *Our pipeline has two stages for each pair of neighboring coarse C and fine F layers.*

$\widehat{M}_i$ and the solution to the previous level M_{i-1}. In this way we compute $M_1, M_2, \ldots, M_k$ in order, ensuring nesting between each subsequent pair. Breaking the problem of nesting many cages into individual subproblems is key to our success as it greatly reduces the complexity of the collision and optimization subproblems.

To simplify notation, from now on, we only consider computing the new positions of a *coarse* mesh C from its original mesh $\widehat{C}$ and the next *finer* output mesh F. Computing new coarse mesh vertex positions $\mathbf{C}$ involves two phases: *flow* of the fine mesh until it is fully inside the coarse mesh, and *re-inflation* of the fine mesh to its original embedding while *pushing* the coarse mesh out of the way (see Figure 8). During the flow, we do not care about the fine mesh's aesthetic surface quality or even whether it self-intersects because we will re-inflate it back to its original positions in the next step.

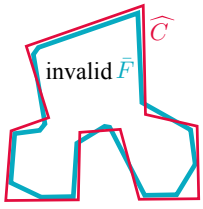
3.1 Flow

The first step of our pipeline is to move vertices $\bar{\mathbf{F}}$ of the fine mesh along a flow that *minimizes total signed distance* to $\widehat{C}$ integrated over all deforming surface points $\bar{\mathbf{p}} \in \bar{F}$ (see Figure 9, top left):

$$\Phi(\bar{\mathbf{F}}) = \int_{\bar{F}} s(\bar{\mathbf{p}}) d(\bar{\mathbf{p}})\, dA, \tag{1}$$

where $d(\bar{\mathbf{p}})$ is the unsigned distance from $\bar{\mathbf{p}}$ to the coarse mesh and $s(\bar{\mathbf{p}})$ modulates by the appropriate sign (negative inside). We minimize Φ by taking small steps opposite its gradient direction for each vertex position $\bar{\mathbf{f}}$ in $\bar{\mathbf{F}}(t)$ as a function of a fictitious *time* t:

$$\frac{\partial \bar{\mathbf{f}}}{\partial t} = -\nabla_{\bar{\mathbf{f}}} \Phi(\bar{\mathbf{F}}) \tag{2}$$



By following this gradient, we *flow* the fine mesh vertices $\bar{\mathbf{F}}(t)$ until all of $\bar{F}$ (not just vertices, see inset) is fully inside the coarse mesh (determined by checking for intersections at each time step).

While Φ is similar to data terms found in iterative closest point (ICP) methods for non-rigid registration [Chang et al. 2010], the sign modulator $s(\bar{\mathbf{p}})$ is an important difference. Minimizing unsigned (positive) distances would flow points toward the surface of the coarse mesh. Instead, by allowing and encouraging negative distances, points flow to the medial axis *within* the coarse mesh.

Since Φ is nonlinear and intractable to compute exactly, we approximate the gradient using numerical quadrature: For each triangle T_i incident on vertices a, b, c, we sample s and d at quadrature points $\bar{\mathbf{p}}_j$ with corresponding weights w_j, $j = 1, \ldots, h$:

$$\Phi(\bar{\mathbf{F}}) = \sum_{i=1}^{m_F} \int_{\bar{\mathbf{p}} \in T_i} s(\bar{\mathbf{p}}) d(\bar{\mathbf{p}})\, dA \quad \approx \sum_{i=1}^{m_F} \sum_{j=1}^{h} w_j s(\bar{\mathbf{p}}_j) d(\bar{\mathbf{p}}_j),$$
$$\bar{\mathbf{p}}_j = \lambda_a \bar{\mathbf{f}}_a + \lambda_b \bar{\mathbf{f}}_b + \lambda_c \bar{\mathbf{f}}_c,$$

where $\lambda_a, \lambda_b, \lambda_c$ are the barycentric coordinates of $\bar{\mathbf{p}}_j$ in T_i and m_F is the number of fine mesh triangles. We use second-order quadrature rules and see diminishing returns with more exact schemes.

The difficulty of differentiating the unsigned distance function $d(\bar{\mathbf{p}}_j)$ remains. To tackle this, we adapt the successful ICP approach of non-rigid registration techniques. Namely, we assume that the closest point $\hat{\mathbf{q}}_j^*$ to each $\bar{\mathbf{p}}_j$ and sign $s(\bar{\mathbf{p}}_j) = s_j^*$ remain constant during each small time step (one could consider modifications common to ICP, e.g. point-to-plane distance, but our goal is not to align the two surfaces, rather to flow one inside the other).

We may now push the gradient through the summation to the terms involving each vertex position $\bar{\mathbf{f}}$:

$$\frac{\partial \bar{\mathbf{f}}}{\partial t} \approx -\sum_{i \in N(\bar{\mathbf{f}})} \sum_{j=1}^{h} w_j s_j^* \nabla_{\bar{\mathbf{f}}} \|\bar{\mathbf{p}}_j - \hat{\mathbf{q}}_j^*\|, \tag{3}$$

where $N(\bar{\mathbf{f}})$ gathers all triangles incident on vertex $\bar{\mathbf{f}}$. Applying the chain rule, and handling the special case where $\bar{\mathbf{p}}_j \approx \hat{\mathbf{q}}_j^*$ (i.e. when our assumption that $s(\bar{\mathbf{p}}_j) = s_j^*$ is invalid), we arrive at

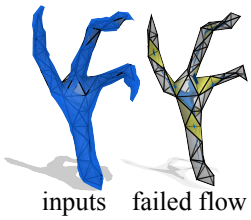
$$\frac{\partial \bar{\mathbf{f}}}{\partial t} \approx -\sum_{i \in N(\bar{\mathbf{f}})} \sum_{j=1}^{h} w_j s_j^* (\nabla_{\bar{\mathbf{f}}} \bar{\mathbf{p}}_j)^{\mathsf{T}} \nabla_{\bar{\mathbf{p}}_j} \|\bar{\mathbf{p}}_j - \hat{\mathbf{q}}_j^*\| \tag{4}$$

$$= -\sum_{i \in N(\bar{\mathbf{f}})} \sum_{j=1}^{h} w_j \lambda_f \mathbf{g}_i, \tag{5}$$

$$\text{where } \mathbf{g}_i = \begin{cases} s_j^* \frac{\hat{\mathbf{p}}_i - \mathbf{q}_i^*}{\|\hat{\mathbf{p}}_i - \mathbf{q}_i^*\|} & \text{if } \|\hat{\mathbf{p}}_i - \mathbf{q}_i^*\| > \epsilon, \\ \mathbf{n}(\mathbf{q}_i^*) & \text{otherwise,} \end{cases} \tag{6}$$

where λ_f is the barycentric coordinate of $\bar{\mathbf{p}}_i$ corresponding to $\bar{\mathbf{f}}$ and $\mathbf{n}(\mathbf{q}_i^*)$ is the unit normal at $\mathbf{q}_i^*$. For $\mathbf{q}_i^*$ near edges and vertices, we use an angle-weighted normal [Baerentzen & Aanaes 2005].

We use a step size $\Delta t \approx 10^{-3}$ (after scaling inputs to unit diameter). After each step we update signs s_i^* and closest points $\mathbf{q}_i^*$ for all quadrature points. We terminate if all signs are negative *and* no intersection exists between $\bar{F}$ and $\widehat{C}$.



Our signed distance flow is not guaranteed to always succeed. Indeed, in difficult cases (e.g. very coarse meshes with very thin features or highly concave vertices) the flow converges without moving the fine mesh fully inside the coarse mesh (see inset). It is possible for all quadrature points on the fine mesh to flow toward the medial axis of the coarse mesh, while stretching triangles through corners leaving intersections (yellow cage facets).

In particularly difficult cases, we propose an additional step: we reverse the picture and expand the *coarse* mesh, flowing it *away* from the current fine mesh along its signed distance field. Contact forces must also be included in this case to ensure the coarse mesh does not flow into a configuration where it self-intersects. Fortunately, in these hard cases only a few expansion steps are typically necessary.

We experimented with an alternative formulation where we *only* consider expansion of the coarse mesh, but this proved problematic.

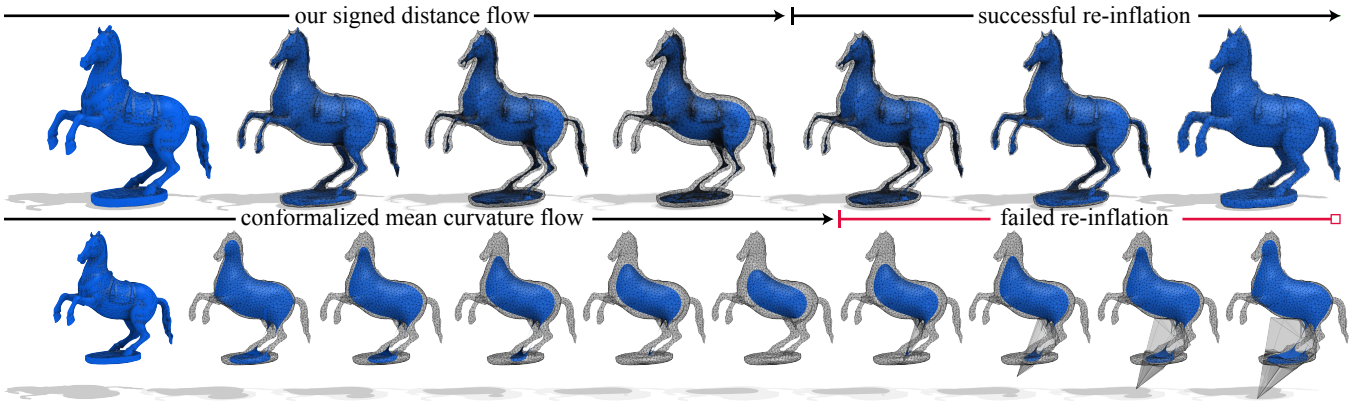


Figure 9: *Our method directly flows the fine mesh (blue) inside the coarse mesh (wireframe). In contrast, curvature flow [Kazhdan et al. 2012] shrinks the fine mesh, but strays outside the coarse mesh. During re-inflation this causes unnecessary collisions, leading to failure.*

Self-contacts may obstruct the flow early on, disrupting further progress (see Figure 6). In addition, the signed distance field outside the fine mesh is, in general, more complicated than inside the coarse mesh: there are more cusps and more variation due to more surface variation in the fine mesh. The coarse mesh inflates too far outside the fine mesh or gets stuck too early, then struggles to *shrink* back into place (particularly around and inside concavities).

Experiments with alternative flows. We experimented with more elegant flows that induce shrinking effects [Taubin 1995; Desbrun et al. 1999; Crane et al. 2013]. Of particular interest were flows that eventually degenerate to the shape's medial axis [Wang & Lee 2008; Au et al. 2008; Tagliasacchi et al. 2012] or to a round point [Kazhdan et al. 2012], but we observed that these often flow the surface outside of its own original volume (much less that of a nearby coarse decimation), particularly for non-convex and high-genus surfaces. Such wandering flows caused unnecessary complications in the re-inflation step we describe next. In Figure 9, we compare to the *conformalized* mean curvature flow of [Kazhdan et al. 2012], a particularly promising method as it is guaranteed to flow sphere-topology surfaces to round points (easily embeddable in the coarse mesh) and that has proven useful in the past [Sacht et al. 2013]. However, for complicated shapes the flow deviates dramatically from the original surface, thus hindering further processing. For high genus shapes it does not resolve intersections upon convergence.

We also experimented with modifications to the surface distance function [Peng et al. 2004], observing that it was not appreciably more robust than the usual L^∞ distance. In fact, simpler experiments of flowing a mesh against itself reveal that designing an inward flow with guarantees is surprisingly difficult. Even picking inward pointing normals at mesh vertices is non-trivial. In fact, the usual uniform-/area-/angle-weighted normals are not guaranteed to point inward. One effective but inelegant normal definition is to tetrahedralize the inner volume and choose a normal pointing toward the center of an incident tetrahedron to each vertex. But even flowing along inward pointing normals at vertices is not guaranteed to nest a mesh inside itself. Counterexamples exist to show that sometimes no inward flow is possible (see Appendix A).

3.2 Re-inflation

After the flow step, $\bar{F}$ is fully inside $\widehat{C}$. We now restore the fine mesh to its original vertex positions $\mathbf{F}$, detecting and resolving collisions with the coarse mesh along the way (see Figure 8).

Jumping directly from $\bar{\mathbf{F}}$ to $\mathbf{F}$ in a single linear step would typically introduce an unwieldy number of simultaneous collisions: too many to disentangle. Fortunately, our previously described signed distance flow provides a meaningful path taking $\bar{F}$ back to F: we simply exactly reverse the motion of the fine mesh, restoring it to its original position in as many iterations as were taken by the flow. As the coarse mesh moves, we detect and respond to would-be intersections between the expanding fine mesh and the current coarse mesh.

We can describe each reverse step in our flow in terms of a displacement per time step, that is, in terms of virtual *velocities*. For ease of notation, we mirror the trajectories of the fine mesh on the time axis so that time continues to point forward (this means, w.l.o.g., $t = 0$ is the moment when the flow with N steps finishes and $t = N\Delta t$ is the moment when the fine mesh returns to its original positions). For the fine mesh, the positions after the next reverse time step are known, and thus so are its velocities:

$$\mathbf{U}_F(t) = \frac{\bar{\mathbf{F}}(t+\Delta t) - \bar{\mathbf{F}}(t)}{\Delta t}, \tag{7}$$

where $\mathbf{U}_F(t) \in \mathbb{R}^{n_F \times 3}$ are instantaneous per-vertex velocities.

The positions of the coarse mesh $\mathbf{C}(t+\Delta t)$—and in turn its similarly defined velocities $\mathbf{U}_C(t)$—are not fixed. In general, there are an infinite number of *feasible* choices of $\mathbf{U}_C(t)$ so that the repositioned coarse mesh $C(t+\Delta t)$ remains free of intersections with itself and with the *re-inflating* fine mesh $\bar{F}(t+\Delta t)$. To regularize this problem, we introduce a generic energy $E(F, \widehat{C}, C)$ measuring the quality of the coarse mesh positions. We optimize this energy to update $\mathbf{C}$ at each reverse time step:

$$\min_{\mathbf{C}(t+\Delta t)} E\left(F, \widehat{C}, \mathbf{C}(t+\Delta t)\right) \qquad \text{subject to:} \tag{8}$$

$$C(s) \text{ does not intersect itself } \forall s \in [t, t+\Delta t], \tag{9}$$

$$C(s) \text{ does not intersect } \bar{F}(s) \forall s \in [t, t+\Delta t], \tag{10}$$

where we are careful to solve the *continuous-time* collision problem rather than only checking for instantaneous collisions at $s = t$ and $s = t + \Delta t$. This ensures the re-inflating fine mesh does not completely *tunnel* through some part of the coarse mesh.

By reformulating our problem in a manner familiar to physical simulation, we may leverage state of the art contact detection (e.g. [Brochu et al. 2012; Wang 2014]) and response methods. Abstractly, we can treat these methods as "black boxes" (velocity filters). We input the fine mesh $\bar{F}(t)$, coarse mesh $C(t)$ and desired velocities $\mathbf{U}_F(t)$ and $\mathbf{U}_C^*(t)$, where $\mathbf{U}_C^*(t)$ is a descent direction minimizing E. The black box outputs new adjusted velocities $\mathbf{U}_F^+(t)$ and $\mathbf{U}_C^+(t)$ satisfying the non-intersection constraints 9-10.

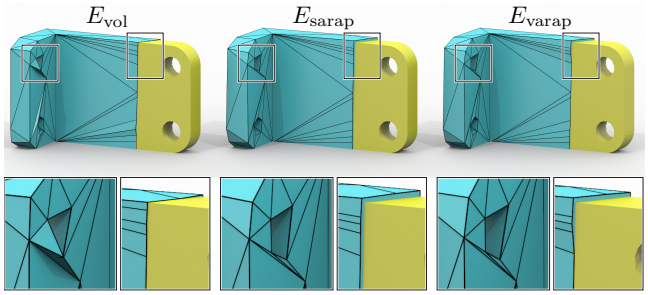


Figure 10: *For a coarse cage around the* Anchor*, optimizing total volume E_{vol} can create zero-volume tents (left inset) but tight fits elsewhere (next inset); surface ARAP E_{sarap} keeps the surface quality, but loosens the fit; volumetric ARAP E_{varap} balances these two goals.*

There remains one interesting twist. Our problem requires the fine mesh to return exactly to its original positions: $\mathbf{U}_F^+(t) = \mathbf{U}_F(t)$. In physically-based simulation parlance, this is tantamount to assigning the fine mesh *infinite mass*. While collision handling is a black box in theory, in practice, the many, many collisions that occur between the coarse and fine meshes, the self-intersecting and degenerate nature of the initial fine mesh, and the fact that the moving mesh has infinite mass, all contribute to a very challenging contact problem beyond the scope of typical cloth and thin shell collision codes. We found two methods that can solve the problem successfully:

First, we adapt the surface tracking method of [Brochu & Bridson 2009] to deal better with infinite masses by removing the *rigid impact zone* phase (similar adaptations of other methods are probably possible, e.g. [Volino & Magnenat-Thalmann 2006; Mueller et al. 2015]).

In cases when this method fails to find a feasible solution, we subdivide Δt recursively. In particularly difficult cases, too many subdivisions are needed, suggesting failure to progress. To handle these hard cases, we fall back on a more robust but slower method: speculative asynchronous contact mechanics [Ainsley et al. 2012]. This method is an extension of the only known method to guarantee intersection prevention and positive progress for *finite mass* objects [Harmon et al. 2009]. Even this algorithm does not guarantee that the collisions can be resolved (in cases where the fine mesh tries to "pinch" the coarse mesh, no solution can possibly exist, see Figure 22) but in practice we see success, albeit at a slow pace.

Choice of energy. We briefly describe the implementations and unique benefits of energies we tested (see Figure 10).

The simplest energies are those measuring vertex displacement in a least squares sense:

$$E_{\text{init}} = \|\mathbf{C}(t+\Delta t) - \widehat{\mathbf{C}}\|_C^2 \tag{11}$$
$$\hookrightarrow \mathbf{U}_C^*(t) \sim \widehat{\mathbf{C}} - \mathbf{C}(t+\Delta t),$$
$$E_{\text{step}} = \|\mathbf{C}(t+\Delta t) - \mathbf{C}(t)\|_C^2 \tag{12}$$
$$\hookrightarrow \mathbf{U}_C^*(t) \sim \mathbf{C}(t) - \mathbf{C}(t+\Delta t),$$

where E_{init} and E_{step} penalize displacement from the coarse mesh's initial positions and from its position at the beginning of the time step, respectively. These energies successfully regularize the space of feasible solutions, but are not representative of energies useful to interesting applications.

For multiresolution hierarchies or collision detection, we propose a volume energy to encourage very tight fitting cages:

$$E_{\text{vol}} = \int_{\text{Vol}(C(t+\Delta t))} 1\, dV = \int_{C(t+\Delta t)} \mathbf{x}\cdot\mathbf{n}\, dA, \tag{13}$$
$$\hookrightarrow \mathbf{U}_C^*(t) \sim -\nabla E_{\text{vol}} = -\mathbf{N}(t+\Delta t),$$

where $\mathbf{N}$ are the area-weighted vertex normals.

During re-inflation, minimizing E_{vol} immediately starts to *shrink-wrap* the coarse mesh $C(t)$ around the expanding fine mesh $\bar{F}(t)$. If $\bar{F}(t)$ needs to expand a significant amount, then this tight shrink-wrap behavior causes unnecessary collisions early in the reverse flow. Therefore, we propose minimizing E_{step} for $t > 0$ in order to find a feasible state before switching to E_{vol} only at $t = N\Delta t$.

The null space of E_{vol} is spanned by all zero-volume meshes, disregarding shape quality or surface area. Shape is retained only by the fine mesh as an obstacle. In some applications, e.g. low-resolution conservative contact replacements or deformation cages, surface appearance or shape-preservation is important. We explored surface-based [Sorkine & Alexa 2007] and volumetric [Chao et al. 2010] forms of as-rigid-as-possible energies (a.k.a. co-rotational elasticity):

$$E_{\text{sarap}} = \sum_{i=1}^{n_C} \underset{\mathbf{R}\in SO(3)}{\text{argmin}} \sum_{j\in N(i)} \|\mathbf{e}_{ij}(t+\Delta t) - \mathbf{R}\hat{\mathbf{e}}_{ij}\|^2 \tag{14}$$
$$\hookrightarrow \mathbf{U}_C^*(t) \sim -\nabla E_{\text{sarap}}$$
$$E_{\text{varap}} = \sum_{T\in\mathcal{T}} \underset{\mathbf{R}\in SO(3)}{\text{argmin}} \sum_{\{ij\}\in T} \|\mathbf{e}_{ij}(t+\Delta t) - \mathbf{R}\hat{\mathbf{e}}_{ij}\|^2 \tag{15}$$
$$\hookrightarrow \mathbf{U}_C^*(t) \sim -\nabla_{\mathbf{C}} E_{\text{varap}}$$

where $\mathbf{e}_{ij}(t+\Delta t)$ and $\hat{\mathbf{e}}_{ij}$ are the edge vectors between vertices i and j of the unknown coarse mesh $C(t+\Delta t)$ and the original coarse mesh $\widehat{C}$, and $\mathcal{T}$ is a list of tetrahedra tessellating $\widehat{C}$. We use TETGEN to create a graded mesh with few auxiliary variables at internal Steiner vertices [Si 2003]. Computing gradients ∇E_{sarap} and ∇E_{varap} involves first optimizing for "best fit rotations" $\mathbf{R}$ via polar decomposition and then computing a sparse matrix product [Chao et al. 2010]. Pure gradient descent of the volumetric ARAP energy is inefficient, as the gradient at interior vertices remains zero until their neighbors move. We accelerate minimization of this energy using a local-global optimization [Sorkine & Alexa 2007]: we update boundary vertices using gradient descent, then optimize for the interior while holding the boundary fixed. This converges quickly when given an initial guess from the previous gradient computation. In any case, our "black box" collision handling dominates computation time, so gradient computation is not a bottleneck.

We continue to minimize the energy even after the fine mesh has returned to its original position, until either $\mathbf{C}(t+\Delta t)$ converges or until bisecting the step length Δt does not decrease the energy. The combination of our energy-minimizing re-inflation and signed-distance flow leads to minimal coarse cage expansion (see Figure 9).

Our method is effectively projected gradient descent. For simple displacement energies, higher-order alternatives do not apply. We experimented with Newton's method for the ARAP energies, but saw little improvement.

4 Results and applications

We implemented a prototype of our method as a serial MATLAB program. We report timings of our unoptimized code for a few representative examples in Table 1 recorded on an iMac Intel Core i7 3.5GHz computer with 8GB memory. As expected the bottleneck is the collision-free re-inflation step. We experimented with a

Model name	Fig. #	#F	k	t_{flow}	t_{re}	Energy
S.W.A.T.	21	9,820	1	5s	269s	Symmetry
Anchor	11	10,778	6	6s	43s	E_{vol}
Warrior	11	26,658	7	2s	86s	E_{varap}
Pelvis	11	40,316	7	11s	460s	E_{vol}
Bunny	13	52,910	7	11s	202s	E_{varap}
Mug	11	74,720	7	7s	54s	E_{vol}
Octopus	11	500,000	11	13s	63s	E_{step}

Table 1: *We report the average time per cage to flow t_{flow}, and to re-inflate t_{re}.*

wide variety of shapes, ranging from CAD models, characters, and scanned objects (see Figure 11). By default we compute layers so that each coarser layer has $2^{-2/3}$ times as many facets as the previous finer layer, a ratio chosen so that resulting tetrahedral meshes will have approximately 8 times fewer elements. For most meshes we compute seven layers, with fewer for lower resolution inputs. In our supplemental material, we attach all input models, corresponding output cages, and a small program to visualize volumetric slices.

Our method is agnostic to the decimator used to create the input meshes $\widehat{M}_1, \ldots, \widehat{M}_k$. In this way we inherit the feature set of the decimator. Figure 12 compares using the regular mesh inducing decimator in [CGAL] (default for all remaining examples) and the feature-adaptive decimator of [Botsch et al. 2002].

To test robustness, we compare computing nested cages on the *Bunny* and the same model corrupted with noise in the normal direction (see Figure 13). The resulting layers tightly hug both shapes.

We also conducted stress tests to evaluate how well our method scales with the number of layers. We nest 25 tightly fitting layers around the *Horse* in Figure 2, and 50 around *Max Planck* in Figure 14. We purposefully continue nesting cages around the the *Gargoyle* in Figure 11, top left, until only eight vertices of an extremely coarse cage remain.

For some applications (e.g. conservative collision detection) there is no need for the output cage to be homeomorphic to the input mesh, and indeed preserving tiny handles unnecessarily increases the complexity of the cage. Our definition of the problem in Section 1 allows the cage to have less handles than the input mesh. Figure 15 illustrates this point, where we generate a cage that is homeomorphic to the input mesh and another cage that has smaller genus.

Constructing our nested cages can be considered expensive *precomputation* for a multiresolution linear system solver. However, once cage meshes are computed and their interiors are meshed with tetrahedra (e.g. using [Si 2003]), the volumetric multigrid solver is sleek and memory efficient. A single multiresolution V-cycle for a Poisson equation with homogeneous Dirichlet boundary conditions inside the volume of the *Octopus* with over seven million vertices takes 1.4 seconds using 2GB max memory. With 14 more V-cycles the solution converges for a total time of 21 seconds (see Figure 16). In contrast, MATLAB's backslash operator thrashes, using over 22GB of memory and finishing in over 16 minutes. CHOLMOD's Cholesky factorization with reordering is mildly better than MATLAB, solving via backsubstitution in 10 minutes, but suffers from similar memory issues during factorization, which takes over an hour using 17GB max memory, due to high fill-in. In terms of precomputation, our time consuming cage computation lives at a much earlier stage than system-matrix factorization: before determining constraints or boundary conditions and before even choosing the particular system being solved. This is even true for inhomogeneous systems

Figure 11: *Each triplet shows: input model, slice through all nested layers, and outermost, coarsest layer. As a stress test, we purposefully continue nesting cages around* Gargoyle *to a very coarse level (top left). The topological holes of the high-genus* Fertility *are maintained across all layers (top right). The deep concavity of the* Mug *does not get smoothed away in coarser levels (bottom right).*

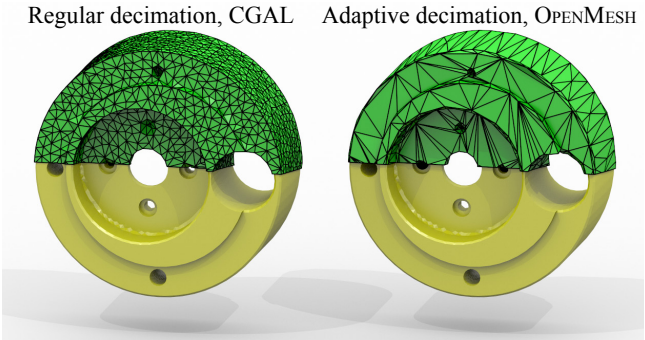


Figure 12: *By post-processing meshes from any existing decimator, our output inherits desired regularity or adaptivity of the decimation.*

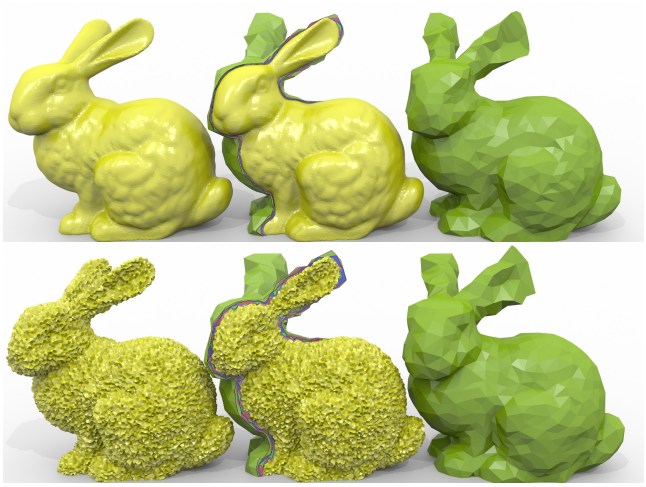


Figure 13: *Noise added in the normal direction to the input bunny does not affect our ability to generate seven quality outer layers.*

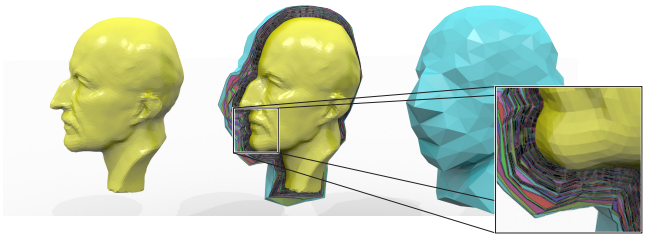


Figure 14: *We fit 50 layers tightly around* Max Planck*'s head.*

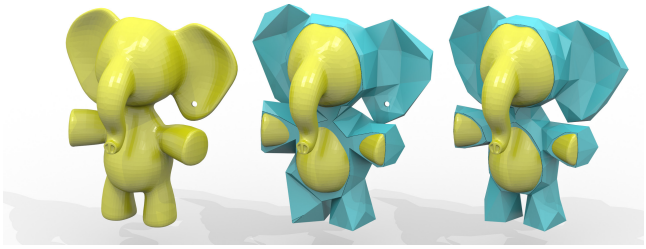


Figure 15: *Depending on the target application, our method generates cages homeomorphic to the input mesh (center) or with handles removed (right).*

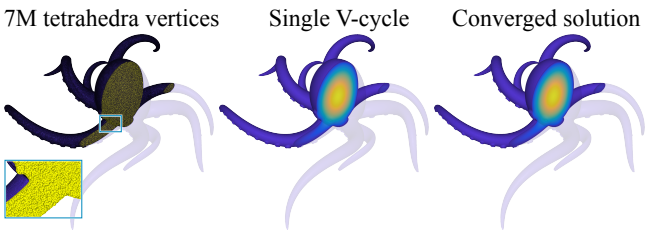


Figure 16: *A single multigrid v-cycle takes 1.4 secs on this 7M-vertex volumetric Poisson equation in the* Octopus*. With 14 more iterations (21 secs) the residual error matches a direct solver's (11 mins, back substitution only).*

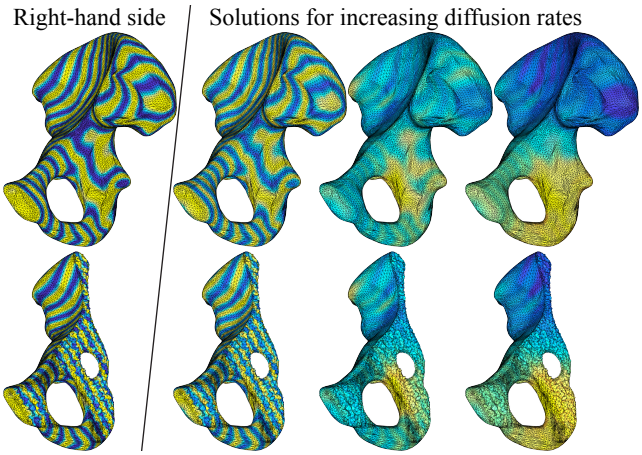
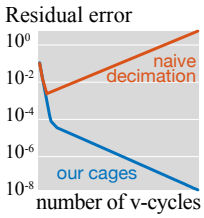


Figure 17: *We solve a diffusion equation* $(\lambda\Delta + I)x = b$ *for various diffusion rates* λ *with our nested cages in a volumetric multiresolution solver (top: surface values via Neumann boundary conditions, bottom: slice through tet-mesh volumes).*

where local metrics vary between solves. In this case, the internal tet-meshing might need to be recomputed (seconds for *Octopus*), but our boundary cages can be reused. Since all fine mesh vertices are inside coarse-mesh tetrahedra, we use linear interpolation for prolongation and its transpose for restriction [Demmel 2004].



As we do not alter the core iterative nature of multiresolution, we benefit from its flexibility. For example, we may quickly change the diffusion rate in a heat equation solved in the volume of the *Pelvis* (see Figure 17). Factorization based solvers, in general, scrap previous precomputation after such a global change to the system matrix. Multiresolution hardly notices, and previous solutions become warm starts. We employ Neumann boundary conditions and notice that naively decimating the input mesh leads to a divergent solver (inset), agreeing with previous analysis that nesting is important for such boundary conditions [Chan et al. 1999]. Because naive decimations do not nest, the prolongation operator must extrapolate for fine mesh vertices lying outside the coarse domain. It *may* be possible to tweak extrapolation parameters to handle these cases with naive decimation, but an automatic method for correcting extrapolation for convergence is not obvious. For comparisons, we tried: linear extrapolation from the nearest tet, constant interpolation of nearest vertex, linear interpolation at the closest point on nearest face. We compare to the most favorable choice.

On less challenging domains, naive decimations can be used, but may require many relaxation (a.k.a. smoothing) iterations on each level of each v-cycle. In Figure 18, we compute smooth geodesics via two Poisson equations in the volume of the *Armadillo* [Crane et al. 2013]. Using our nested cages, the multiresolution solver converges independent of the number of relaxation iterations used (typically fewer relaxation iterations and more v-cycles is preferable). In contrast, multiresolution using naive non-nested decimations converge when using a large number of relaxation iterations, and then at a rate equivalent to single relaxation iteration with our meshes.

A single enclosing cage is useful for creating a lower dimensional volumetric domain for elastic simulation of an input model that is either too high resolution or too complex due to meshing imperfections [Xu & Barbič 2014]. In Figure 19, we compare to extracting the

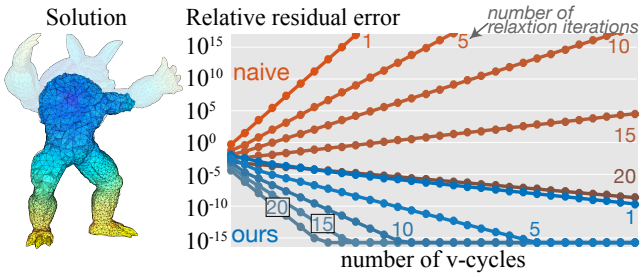


Figure 18: *We solve for smooth geodesics over a volumetric tetrahedral mesh inside the* Armadillo. *Using naive overlapping and shrinking decimations leads to divergence unless a very large number of relaxation iterations is used. Ours is always convergent.*

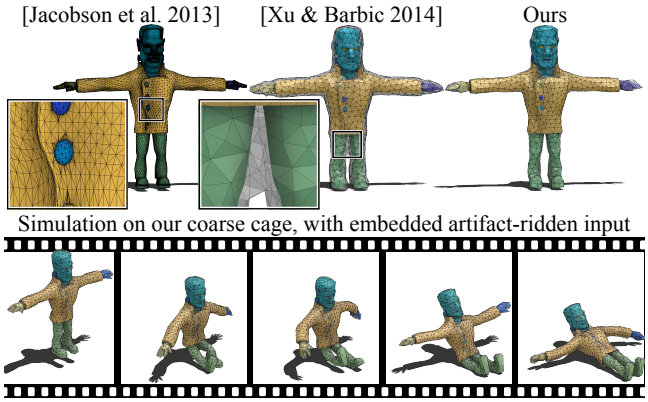


Figure 19: *Extracting an outer hull with Jacobson et al [2013] fails to coarsen the domain (top left). Contouring a distance field achieves nesting at a large iso-level but fuses the legs (top center). Our coarse cage fits the input tightly and provides a reduced domain for real-time physics.*

outer hull of the multi-component and self-intersecting *Frankenstein* using [Jacobson et al. 2013]. While technically a "perfectly tight fit", this cage fails to coarsen the domain and meshing near intersections creates sliver triangles problematic for numerics. We also compare to signed-distance field contouring [Xu & Barbič 2014], which creates a loose fit joining the legs near the knees. With the same vertex count, our cage is a tight fit making it especially suitable for elastic simulation with collision handling. The deformation of the volume inside the coarse cage is then propagated via linear interpolation to the embedded mesh (see also accompanying video).

Another application area for nested cages is collision detection for rigid objects: if an object does not collide against a surface enclosing a second object, this certifies that the two objects do not collide either. Bounding sphere, cubes, and higher-degree polytopes are often used to quick-reject candidate collisions for this reason, but these convex cages lose effectiveness when objects are concave or have holes and cavities. On the other hand, a coarse nested cage is ideal for this purpose, since the nesting property guarantees correctness and the tight fit allows the coarse cage to efficiently reject nearly all false positives during collision detection. As a proof of concept, we simulated dropping eighteen instances of the octopus mesh (Figure 11, bottom-left) one by one into a narrow tank, where they collide with each other and the tank walls (with coefficient of restitution 0.99) using continuous-time collision detection (see inset). We compared the performance of two different broad phase strategies for collision detection:

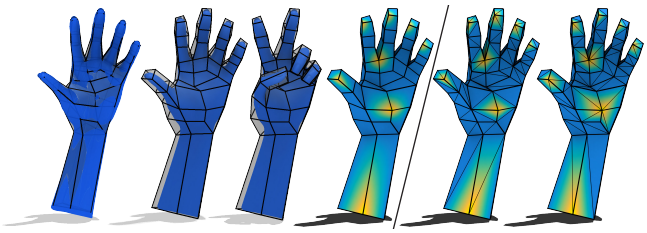


Figure 20: *Left to right: a quad mesh is quickly sketched atop the* Hand *and our pipeline moves it outside the input while planarizing quads. A cage-based deformation is applied via harmonic coordinates computed over the volume inside the planar-quad polyhedron (a few coordinates visualized in pseudocolor). These weights are smooth inside each quad. In contrast, triangulating the quads would lead to non-smooth, meshing-dependent values in each quad (two alternative triangulations).*

in the first, bounding cubes around swept spacetime volumes are used to prune distant pairs of objects from consideration, then kDOP bounding volume hierarchies are built to find candidate colliding vertex-face pairs, which are then passed to the continuous-time narrow phase. The second strategy is the same, except that we first check if the octopus's coarse cage is colliding before building the BVH on the octopus itself. We found that the latter strategy is $\sim 8\times$ faster, for intuitive reasons: the octopus's many protruding arms causes it to easily nestle near other copies of itself, so that their convex hulls overlap but our tight, coarse cages do not. Nesting is applicable for rigid or nearly rigid objects, but it is not obvious how to track cages along with deformable bodies, unless, for example, deformations could be precomputed.

The ability to customize our optimization energy enables not just better cages, but also better generalized barycentric coordinates. In particular, harmonic coordinates are defined for arbitrary polyhedra, yet most works implement them inside triangle-mesh cages only [Joshi et al. 2007]. To utilize popular quad-dominant meshes as cages, all faces must be outside the input model and *planar*. Since such cages are difficult to model manually, many implementations simply triangulate high-order facets [Joshi et al. 2007]. We complement the recent sketch-based quad-meshing tool [Takayama et al. 2013], post-processing its output to enclose the input model and minimize a planarization energy [Poranne et al. 2013]. Quad-dominant cages are easier to control as their visualization is less cluttered (see Figure 20). More importantly, harmonic coordinates constructed (via their recursive definition) on the planar-quad polyhedron are also higher quality: coordinates are smooth functions inside each quad. In contrast, coordinates of a triangulated cage would depend heavily on the choice of diagonals splitting each quad.

Generalizations beyond watertight meshes. Though not strictly meeting our input criteria, we apply our method to polygon soups. In Figure 21, we again adapt the optimization energy, this time to maintain the reflectional symmetry of the coarse input cage. The input polygon soup *S.W.A.T. man* is riddled with meshing artifacts, but we still flow it inside. Though details on the expanding input mesh are asymmetric (see hip pockets or hands), the energy minimization keeps the cage symmetric.

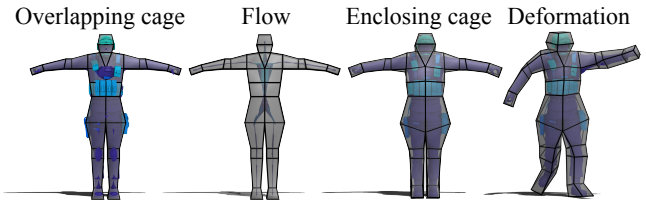


Figure 21: S.W.A.T. man *is a polygon soup with 2806 intersecting triangle pairs, 24 non-manifold edges, 51 boundary loops and 51 components. Our shrinking flow is robust to such artifacts. Once inside the coarse, overlapping input cage, we re-inflate it and produce a fully exterior cage used to deform the embedded model.*

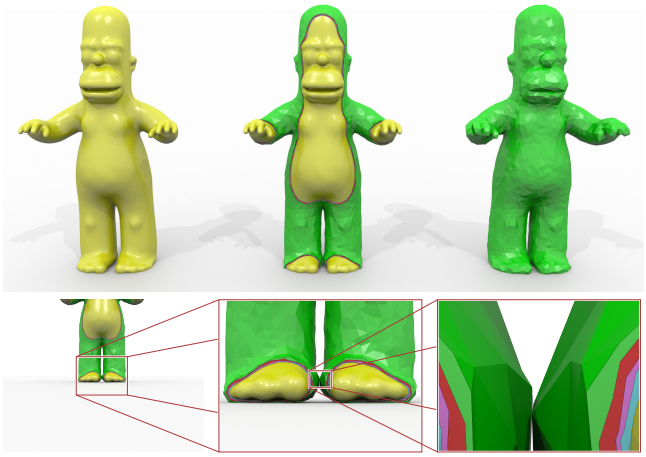


Figure 22: *If an expanding coarse mesh collides with itself (green), it creates a* pinch *preventing processing of further coarser layers.*

5 Limitations and future work

We plan to optimize the performance of both the signed distance field flow and the re-inflation steps. Collision detection and response dominates running time, and our prototype naively recomputes acceleration data-structures rather than updating them continuously.

Our insight to break the multi-layer nesting problem into pairwise subproblems ensures tractability, but in some cases leads to converging at an "artificial local minimum." If a coarse cage collides with itself during inflation then it may create a pinch that blocks inflation of subsequent coarse layers (see Figure 22). One solution is to iterate through the fine layers to make sufficient room in these problem areas, but defining this relaxation direction is not obvious.

In cases where the input coarse cage begins too far away from the fine mesh, the signed-distance flow will fail: for example, by flowing vertices of a triangle into opposite parts of the coarse mesh. Adding a small amount of smoothing on the flowing fine mesh or expanding the coarse mesh alleviates some of these problems, but a general solution is elusive. The correct assignment seems related to correctly matching medial axes of both meshes: perhaps an avenue of future improvement.

We believe the performance of our multigrid solver could be further improved by parameter tuning and experimenting with different coarsening gradations. We would also like to consider using our meshes to build multigrid preconditioners for conjugate gradient solvers. We expect that higher order PDEs with more involved boundary conditions will receive an even greater benefit from our nested cages.

It would be interesting to analyze formally the convergence of our nested cages along the lines of [Chan & Zou 1996] who consider the then-available non-nested hierarchies.

Our cages are designed for *volumetric* multigrid solvers, and are not immediately applicable to *surface-based* multiresolution problems (cf. [Aksoylu et al. 2005; Chuang et al. 2009]). Whether nesting is at all useful for surface problems remains an open question.

In conclusion, nested cages prove to be a powerful tool in a variety of applications. Our signed-distance flow consistently finds initial feasible states for our constraint-based optimization. By leveraging state-of-the-art collision handling tools from physically based simulation, we are able to generate cages that in turn enable faster physical simulations, more-efficient linear system solvers and better real-time deformation user interfaces. We hope that our algorithm's success encourages more multiresolution volumetric methods using unstructured meshes in geometry processing, computer graphics, and beyond.

Acknowledgements

Humanoid models courtesy Ilya Baran were initially created using Cosmic Blobs(R) software developed by Dassault Systèmes SolidWorks Corp. Medical models courtesy of Muhibur Rasheed. We thank: Derek Bradley, Keenan Crane, Eitan Grinspun, and Daniele Panozzo, for illuminating discussions; Eric Price, for brainstorming the NP completeness proof; Henrique Maia, Papoj Thamjaroenporn, and Sarah Abraham, for proofreading; and Peter Schroeder, Richard Kenyon, Alexander I. Bobenko, Helmut Pottmann, and Johannes Wallner for organizing the DDG Oberwolfach and Seggau Geometry workshops. The Columbia Computer Graphics Group is supported by the NSF, Intel, The Walt Disney Company, and Autodesk. Funded in part by NSF grant DMS-1304211. We thank the Visgraf Lab and IMPA, for the technical support and software resources, CNPq, for the first author's PhD fellowship, and CAPES and FAPESC, for supporting the presentation of this work at SIGGRAPH Asia 2015.

References

ADAMS, M., AND DEMMEL, J. W. 1999. Parallel multigrid solver for 3d unstructured finite element problems. In *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*.

AINSLEY, S., VOUGA, E., GRINSPUN, E., AND TAMSTORF, R. 2012. Speculative parallel asynchronous contact mechanics. *ACM Trans. Graph. 31*, 6.

AKSOYLU, B., KHODAKOVSKY, A., AND SCHRÖDER, P. 2005. Multilevel Solvers for Unstructured Surface Meshes. *SIAM Journal on Scientific Computing 26*, 4, 1146.

ATTENE, M. 2010. A lightweight approach to repairing digitized polygon meshes. *The Visual Computer 26*, 11, 1393–1406.

AU, O. K.-C., TAI, C.-L., CHU, H.-K., COHEN-OR, D., AND LEE, T.-Y. 2008. Skeleton extraction by mesh contraction. *ACM Trans. Graph. 27*, 3.

BAERENTZEN, J. A., AND AANAES, H. 2005. Signed distance computation using the angle weighted pseudonormal. *Trans. Vis. & Comp. Graphics 11*, 3 (May).

BARAFF, D., WITKIN, A., AND KASS, M. 2003. Untangling cloth. *ACM Trans. Graph. 22*.

BEN-CHEN, M., WEBER, O., AND GOTSMAN, C. 2009. Spatial deformation transfer. In *Proc. SCA*, 67–74.

BEN-CHEN, M., WEBER, O., AND GOTSMAN, C. 2009. Variational harmonic maps for space deformation. *ACM Trans. Graph. 28*, 3.

BOTSCH, M., STEINBERG, S., BISCHOFF, S., AND KOBBELT, L. 2002. OpenMesh – a generic and efficient polygon mesh data structure. In *OpenSG Symposium*.

BROCHU, T., AND BRIDSON, R. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Sci. Comp. 31*, 4.

BROCHU, T., EDWARDS, E., AND BRIDSON, R. 2012. Efficient geometrically exact continuous collision detection. *ACM Trans. Graph. 31*, 4.

BRUNE, P. R., KNEPLEY, M. G., AND SCOTT, L. R. 2011. Unstructured geometric multigrid in two and three dimensions on complex and graded meshes. *CoRR abs/1104.0261*.

CAMPEN, M., AND KOBBELT, L. 2010. Polygonal Boundary Evaluation of Minkowski Sums and Swept Volumes. *Comput. Graph. Forum 29*, 5, 1613–1622.

CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

CHAN, T. F., AND WAN, W. 2000. Robust multigrid methods for nonsmooth coefficient elliptic linear systems. *J. Comput. Appl. Math. 123*, 1-2.

CHAN, T. F., AND ZOU, J. 1996. A convergence theory of multilevel additive schwarz methods on unstructured meshes. *Numerical Algorithms 13*, 2, 365–398.

CHAN, T. F., SMITH, B., AND ZOU, J. 1996. Overlapping schwarz methods on unstructured meshes using non-matching coarse grids. *Numer. Math 73*, 149–167.

CHAN, T. F., GO, S., AND ZOU, J. 1999. Boundary treatments for multilevel methods on unstructured meshes. *SIAM Journal on Scientific Computing 21*, 1, 46–66.

CHANG, W., LI, H., MITRA, N. J., PAULY, M., AND WAND, M. 2010. Geometric registration for deformable shapes. In *Eurographics 2010: Tutorial Notes*.

CHAO, I., PINKALL, U., SANAN, P., AND SCHRÖDER, P. 2010. A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4.

CHUANG, M., LUO, L., BROWN, B. J., RUSINKIEWICZ, S., AND KAZHDAN, M. 2009. Estimating the laplace-beltrami operator by restricting 3d functions. In *Proc. SGP*, 1475–1484.

CRANE, K., PINKALL, U., AND SCHRÖDER, P. 2013. Robust fairing via conformal curvature flow. *ACM Trans. Graph. 32*, 4.

DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. H. 2001. Dynamic real-time deformations using space & time adaptive sampling. *Proceedings of SIGGRAPH 2001*, 31–36.

DEMMEL, J., 2004. Multigrid overview. `http://www.cs.berkeley.edu/~demmel`.

DENG, Z.-J., LUO, X.-N., AND MIAO, X.-P. 2011. Automatic cage building with quadric error metrics. *Comp. Sci. & Tech. 26*, 3.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proccedings of ACM SIGGRAPH*, 317–324.

DICK, C., GEORGII, J., AND WESTERMANN, R. 2011. A hexahedral multigrid approach for simulating cuts in deformable objects. *Trans. Vis. & Comp. Graphics 17*, 11.

DICKOPF, T. 2010. *Multilevel methods based on non-nested meshes*. PhD thesis, Universität Bonn.

EPPSTEIN, D., 2015. When is it possible to "shrink" a polyhedron? MathOverflow. `http://mathoverflow.net/q/206750` (version: 2015-05-16).

FALGOUT, R. 2006. An introduction to algebraic multigrid computing. *Computing in Science Engineering* (Nov).

FENG, Y., PERIĆ, D., AND OWEN, D. 1997. A non-nested galerkin multi-grid method for solving linear and nonlinear solid mechanics problems. *Comp. methods in applied mech. & eng. 144*, 3.

FERSTL, F., WESTERMANN, R., AND DICK, C. 2014. Large-scale liquid simulation on adaptive hexahedral grids. *Trans. Vis. & Comp. Graphics 20*, 10.

FISH, J., PANDHEERADI, M., AND BELSKY, V. 1995. An efficient multilevel solution scheme for large scale non-linear systems. *Int. Journal for Numerical Methods in Eng. 38*, 10, 1597–1610.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH*.

GUILLARD, H. 1993. Node-nested multi-grid method with Delaunay coarsening. Research Report RR-1898.

GUMHOLD, S., BORODIN, P., AND KLEIN, R. 2003. Intersection free simplification. *IJSM*, 155–176.

HARMON, D., VOUGA, E., SMITH, B., TAMSTORF, R., AND GRINSPUN, E. 2009. Asynchronous contact mechanics. *ACM Trans. Graph. 28*, 3.

HARMON, D., PANOZZO, D., SORKINE, O., AND ZORIN, D. 2011. Interference aware geometric modeling. *ACM Trans. Graph. 30*, 6.

HOPPE, H. 1996. Progressive meshes. In *Proc. SIGGRAPH*.

JACOBSON, A., KAVAN, L., AND SORKINE-HORNUNG, O. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph. 32*, 4.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3, 71.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proc. SGP*, 61–70.

KAZHDAN, M., SOLOMON, J., AND BEN-CHEN, M. 2012. Can mean-curvature flow be modified to be non-singular? *Comput. Graph. Forum 31*, 5.

KRISHNAN, D., FATTAL, R., AND SZELISKI, R. 2013. Efficient preconditioning of laplacian matrices for computer graphics. *ACM Trans. Graph. 32*, 4.

MCADAMS, A., ZHU, Y., SELLE, A., EMPEY, M., TAMSTORF, R., TERAN, J., AND SIFAKIS, E. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph. 30* (Aug.), 37:1–37:12.

MEHRA, R., ZHOU, Q., LONG, J., SHEFFER, A., GOOCH, A., AND MITRA, N. J. 2009. Abstraction of man-made shapes. *ACM Trans. Graph. 28*, 5.

MELAX, S. 1998. A simple, fast, and effective polygon reduction algorithm. *Game Developer Magazine*, 44–49.

MUELLER, M., CHENTANEZ, N., KIM, T.-Y., AND MACKLIN, M. 2015. Air meshes for robust collision handling. *ACM Trans. Graph. 34*, 4.

NESME, M., KRY, P. G., JEŘÁBKOVÁ, L., AND FAURE, F. 2009. Preserving topology and elasticity for embedded deformable models. *ACM Trans. Graph. 28*, 3.

OTADUY, M. A., AND LIN, M. C. 2003. CLODs: Dual hierarchies for multiresolution collision detection. In *Proc. SGP*.

OTADUY, M. A., AND LIN, M. C. 2003. Sensation preserving simplification for haptic rendering. *ACM Trans. Graph. 22*.

PENG, J., KRISTJANSSON, D., AND ZORIN, D. 2004. Interactive modeling of topologically complex geometric detail. *ACM Trans. Graph. 23*, 3, 635–643.

PETERHANS, C. 2012. *Interactive Surface Reconstruction*. Master's thesis, ETH Zurich.

PLATIS, N., AND THEOHARIS, T. 2003. Progressive hulls for intersection applications. *Comput. Graph. Forum 22*, 2.

PORANNE, R., OVREIU, E., AND GOTSMAN, C. 2013. Interactive planarization and optimization of 3d meshes. In *Comput. Graph. Forum*, vol. 32, 152–163.

RUGE, J., AND STÜBEN, K. 1987. Algebraic multigrid. *Multigrid methods 3*.

SACHT, L., JACOBSON, A., PANOZZO, D., SCHÜLLER, C., AND SORKINE-HORNUNG, O. 2013. Consistent volumetric discretizations inside self-intersecting surfaces. In *Proc. SGP*.

SANDER, P. V., GU, X., GORTLER, S. J., HOPPE, H., AND SNYDER, J. 2000. Silhouette clipping. In *Proc. SIGGRAPH*.

SHEN, C., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2004. Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. Graph. 23*, 3, 896–904.

SI, H., 2003. TETGEN: A 3D delaunay tetrahedral mesh generator. `http://tetgen.berlios.de`.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proc. SGP*, 109–116.

SÝKORA, D., DINGLIANA, J., AND COLLINS, S. 2009. As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proc. NPAR*.

TAGLIASACCHI, A., ALHASHIM, I., OLSON, M., AND ZHANG, H. 2012. Mean curvature skeletons. *Comput. Graph. Forum 31*, 5.

TAKAYAMA, K., PANOZZO, D., SORKINE-HORNUNG, A., AND SORKINE-HORNUNG, O. 2013. Sketch-based generation and editing of quad meshes. *ACM Trans. Graph. 32*, 4.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH*, 351–358.

TERAN, J., SIFAKIS, E., BLEMKER, S. S., NG-THOW-HING, V., LAU, C., AND FEDKIW, R. 2005. Creating and simulating skeletal muscle from the visible human data set. *Trans. Vis. & Comp. Graphics 11*, 3.

VOLINO, P., AND MAGNENAT-THALMANN, N. 2006. Resolving surface collisions through intersection contour minimization. *ACM Trans. Graph. 25*, 3.

VOUGA, E., SACHT, L., AND JACOBSON, A. 2015. Polyhedral nesting is NP-complete. Tech. rep., in supplementary materials.

WANG, Y.-S., AND LEE, T.-Y. 2008. Curve-skeleton extraction using iterative least squares optimization. *Trans. Vis. & Comp. Graphics 14*, 4.

WANG, H., SIDOROV, K. A., SANDILANDS, P., AND KOMURA, T. 2013. Harmonic parameterization by electrostatics. *ACM Trans. Graph. 32*, 5.

WANG, H. 2014. Defending continuous collision detection against errors. *ACM Trans. Graph. 33*, 4.

WICKER, M., LANKER, H., AND GROSS, M. 2006. Untangling cloth with boundaries. In *Proc. VMV*.

XIAN, C., LIN, H., AND GAO, S. 2009. Automatic generation of coarse bounding cages from dense meshes. In *Proc. SMA*.

XIAN, C., LIN, H., AND GAO, S. 2012. Automatic cage generation by improved OBBs for mesh deformation. *The Visual Computer*.

XIAN, C., ZHANG, T., AND GAO, S. 2013. Semantic cage generation for fe mesh editing. In *CADG*.

XIAN, C., LI, G., AND XIONG, Y. 2015. Efficient and effective cage generation by region decomposition. *Computer Animation and Virtual Worlds 26*, 2, 173–184.

XU, H., AND BARBIČ, J. 2014. Signed distance fields for polygon soup meshes. *Graphics Interface 2014*.

YE, J., AND ZHAO, J. 2012. The intersection contour minimization method for untangling oriented deformable surfaces. In *Proc. SCA*.

Appendix A: Flow counterexample

There exist polyhedra that cannot be continuously flowed inside themselves [Eppstein 2015]. Consider the surface constructed in Figure 23, right. Though inward pointing normals can be defined at all vertices, any non-negative (even infinitesimal) inward movement of the vertices will introduce "edge-edge" collisions with the original surface. To see this, consider the set of points inside the polyhedron that can be "seen" by the central vertex c of the star. This set changes discontinuously for any motion of c: no matter which direction is chosen to flow c, there exists some neighbor n such that c loses sight of not only n, but also an entire neighborhood around n. Therefore there exists no inward flow of all of the vertices where c maintains sight with all of its neighbors.

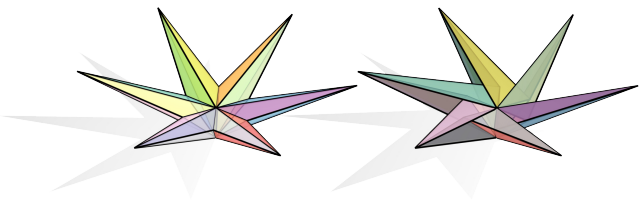


Figure 23: *Take every other vertex around the star surface on the left and rotate it about the origin to create the "origami pinwheel" on the right. Vertices of this new surface cannot be continuously flowed inward without creating intersections with the original surface.*

Appendix B: Pseudocode

Algorithm 2: Shrink$(\widehat{C}, F) \rightarrow H$

Inputs:
- $\widehat{C}$ Initial coarse mesh (will remain constant)
- F Initial fine mesh (possibly overlapping with $\widehat{C}$)

Outputs:
- H Fine mesh history: $H \leftarrow \{\bar{\mathbf{F}}(0), \bar{\mathbf{F}}(\Delta t), \ldots, \bar{\mathbf{F}}(N\Delta t)\}$

begin
- $H \leftarrow \{F\}$ *// Initialize history with input fine mesh*
- **while** $\widehat{C}$ does not nest H.last **do**
  - $\nabla\Phi \leftarrow 0$ *// Initialize gradients to zero*
  - **for** each quadrature point $\bar{\mathbf{p}}_i$ on H.last **do**
    - $\mathbf{q} \leftarrow$ closest point to $\bar{\mathbf{p}}_i$ on $\widehat{C}$
    - $s \leftarrow \begin{cases} 1 & \text{if } \bar{\mathbf{p}}_i \text{ is outside } \widehat{C} \\ -1 & \text{otherwise} \end{cases}$ *// Distance sign*
    - **if** $\|\bar{\mathbf{p}}_i - \mathbf{q}\| >$ 1e-5 **then**
      - $\mathbf{g} \leftarrow s(\bar{\mathbf{p}}_i - \mathbf{q})/\|\bar{\mathbf{p}}_i - \mathbf{q}\|$
    - **else**
      - */* If too close use normal [**Baerentzen & Aanaes 2005**] */*
      - $\mathbf{g} \leftarrow \mathbf{n}(\mathbf{q})$
    - **for** each vertex j in H.last **do**
      - */* w_i is the weight of the quadrature point and λ_{ij} is the hat function of vertex j evaluated at quadrature point $\bar{\mathbf{p}}_i$ */*
      - $\nabla\Phi(j) \leftarrow \Phi(j) + w_i\lambda_{ij}\mathbf{g}$
  - $\Delta t \leftarrow$ 1e-3 *// Default time step*
  - H.push(H.last $- \Delta t \cdot \nabla\Phi$)

Algorithm 3: Reinflate$(H, \widehat{C}, \text{Energy}) \rightarrow C$

Inputs:
- H Fine mesh history: $H \leftarrow \{\bar{\mathbf{F}}(0), \bar{\mathbf{F}}(\Delta t), \ldots, \bar{\mathbf{F}}(N\Delta t)\}$
- $\widehat{C}$ Initial coarse mesh
- Energy Function object for re-inflation energy and gradient

Outputs:
- C Final nested cage

begin
- $F_0 \leftarrow H$.first *// Initial fine mesh*
- $F \leftarrow H$.pop *// Shrunken fine mesh*
- $C \leftarrow \widehat{C}$ *// Initialize output cage*
- **while** H is not empty **do**
  - $\beta \leftarrow \beta_{\text{init}}(=$ 1e-2) *// Initial step size*
  - $E_{\text{min}} \leftarrow \infty$ *// Initial minimum energy*
  - $\mathbf{U}_F \leftarrow H$.pop $- F$ *// Desired velocities for fine mesh*
  - **repeat**
    - $\mathbf{U}_C \leftarrow -\beta\,\nabla\text{Energy}(F_0, C, \widehat{C})$ *// —"— coarse mesh*
    - */* Filter velocities, e.g. modified [**Brochu & Bridson 2009**] or [**Ainsley et al. 2012**] */*
    - $\mathbf{U}_C \leftarrow \text{Filter}(F, \mathbf{U}_F, C, \mathbf{U}_C)$
    - **if** Energy$(C + \mathbf{U}_C) > E_{\text{min}}$ **then**
      - $\beta \leftarrow \beta/2$ *// Decrease step size*
      - **if** $\beta < \beta_{min}(=$ 1e-3) **then break** *// No progress*
    - **else**
      - $C \leftarrow C + \mathbf{U}_C$ *// Step coarse mesh*
      - $E_{\text{min}} \leftarrow$ Energy(C) *// Update energy*
      - $\beta \leftarrow 1.1\beta$ *// Slightly increase step size*
    - **if** $\|\mathbf{U}_C\| < \Delta C(=$ 1e-5) **then break** *// "Converged"*
  - $F \leftarrow F + \mathbf{U}_F$ *// Step fine mesh*