Nested Cages
Leonardo Sacht\textsuperscript{1}  
Etienne Vouga\textsuperscript{2}  
Alec Jacobson\textsuperscript{3}

\textsuperscript{1}Universidade Federal de Santa Catarina  
\textsuperscript{2}University of Texas at Austin  
\textsuperscript{3}Columbia University

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 \textit{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].

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{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.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{25 increasingly coarse cages nested around a Horse. Our tight nesting property is robust even for a large number of cages.}
\end{figure}
This problem in its full generality is intractable. For single polygons $F$, this definition extends naturally to when $F$ surrounds $C$ 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.

Given an embedded polyhedron $F$, a polyhedron $\hat{C}$, and a fitness function $E$, the goal is to minimize $E(F, \hat{C}, C)$ over all embeddings $C$ of $\hat{C}$ 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 $\hat{C}$ in a way that nests $F$ (see Figure 3). The decision problem of whether any nesting of $\hat{C}$ around $F$ exists can be shown to be NP-complete [Vouga et al. 2015]. Fortunately, in practice $\hat{C}$ and $F$ are not completely arbitrary; any reasonable decimation scheme will keep $\hat{C}$ 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 $\hat{C}$ and fitness function $E$. Briefly, the method first flows $F$ inside $\hat{C}$ along $\hat{C}$’s signed distance field; once inside, we rewind the flow while deforming $\hat{C}$ 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 $\hat{C}$, rather than accepting it as input from the user, the problem becomes substantially easier, and has been extensively studied:

Bounding polytopes. Most naïvely, one can simply take $\hat{C}$ to be a canonical bounding polytope (e.g. box, KDOP, convex hull). An extension would be to “shrink-wrap” the bounding polytope around $\hat{F}$ 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 $\hat{F}$—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 $\hat{F}$ as an implicit surface and contouring level sets is a popular method of creating offset surfaces that nest $\hat{F}$, 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 $\hat{F}$ 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 et al [2004] iteratively refine a moving least squares iso-surface to enclose an existing model tightly via a global scaling parameter.

![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.](image)

![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.](image)
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 \( \hat{F} \) 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 [Platt & Theoharis 2003]. Although the vertices of these “progressive hulls” are guaranteed to lie outside of \( \hat{F} \), edges and faces of \( \hat{C} \) might still intersect \( \hat{F} \) (and \( \hat{C} \) 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 temporal 3D tetrahedral meshing in tight regions near overlaps and does not consider face- or edge-intersections with the input model.

**Voxelization** of \( \hat{F} \) 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). Naïve 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;
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 [Deburne 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, [Gunhold 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]. Otaduy 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 groundbreaking 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 \( \tilde{M}_0, \tilde{M}_1, \ldots, \tilde{M}_k \). Each mesh \( \tilde{M}_i \) has a corresponding list of initial vertex positions \( \tilde{M}_i \in \mathbb{R}^{n_i \times 3} \) and list of triangle indices \( T_i \in \{1, \ldots, n_i\}^{n_i \times 3} \). In a typical scenario, \( \tilde{M}_0 \) is a high-resolution original mesh and \( \tilde{M}_1, \ldots, \tilde{M}_k \) are decimations of decreasing resolution; the method of decimation is unimportant so long as the input meshes all approximate \( \tilde{M}_0 \) 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]).

The output of our method is a new sequence of \( k \) lists of vertex positions \( M_1, \ldots, M_k \) such that for \( i = 1, \ldots k \) each \( M_i = (T_i, \tilde{M}_i) \) is a deformed mesh whose surface nests \( M_{i-1} \) (with \( M_0 = \tilde{M}_0 \)). 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 \( \tilde{M}_0, \tilde{M}_1, \ldots, \tilde{M}_k \) (that 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 \( M_{i-1} \) 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**: \( \text{nested_cages}(M_0, L, \text{Dec}, \text{Energy}) \rightarrow M_1, \ldots, M_k \)

**Inputs:**
- \( M_0 \) Initial high-res mesh, vertices \( M_0 \), faces \( T_0 \)
- \( L \) \( k \)-long list of desired mesh resolutions
- \( \text{Dec} \) Function object for black-box decimator
- \( \text{Energy} \) Function object for re-inflation energy and gradient

**Outputs:**
- \( M_1, \ldots, M_k \) \( k \)-long list of nested cages

**begin**
- for \( i \in \{1, \ldots, k\} \) do
  - /* Decimate from previous layer or input mesh */
  - \( \tilde{M}_i \leftarrow \text{Dec} \left( M_{i-1}, \tilde{M}_{i-1}, L(i) \right) \)
  - \( F \leftarrow M_{i-1}, \tilde{C} \leftarrow \tilde{M}_i \) // Rename coarse and fine meshes
  - \( H \leftarrow \text{Shrink}(\tilde{C}, F) \) // history of shrinking fine mesh
  - /* Reverse history, then \( M_i \) will nest \( M_{i-1} \)*/
  - \( M_i \leftarrow \text{ReInflate}(H, \tilde{C}, \text{Energy}) \)

**end**

Our method operates recursively on two meshes of the sequence at a time: we compute \( M_i \) by considering only its original embedding.
Our pipeline has two stages for each pair of neighboring coarse $C$ and fine $F$ layers.

$\hat{M}_i$ 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 $\hat{C}$ and the next finer output mesh $F$. Computing new coarse mesh vertex positions $C$ 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 $F$ of the fine mesh along a flow that minimizes total signed distance to $\hat{C}$ integrated over all deforming surface points $p \in F$ (see Figure 9, top left):

$$\Phi(F) = \int_F s(p)d(p)\,dA, \quad (1)$$

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

$$\frac{\partial \Phi}{\partial t} = -\nabla_F \Phi(F) \quad (2)$$

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

While $\Phi$ is similar to data terms found in iterative closest point (ICP) methods for non-rigid registration [Chang et al. 2010], the sign modulator $s(p)$ 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 $\Phi$ 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 $p_j$ with corresponding weights $w_j, j = 1, \ldots, h$:

$$\Phi(F) = \sum_{i=1}^{m_F} \sum_{p \in T_i} s(p)d(p)\,dA \approx \sum_{i=1}^{m_F} \sum_{j=1}^{h} w_j s(p_j)d(p_j), \quad (3)$$

$$p_j = \lambda_a \hat{F}_a + \lambda_b \hat{F}_b + \lambda_c \hat{F}_c,$$

where $\lambda_a, \lambda_b, \lambda_c$ are the barycentric coordinates of $p_j$ 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 $d(p_j)$ remains. To tackle this, we adapt the successful ICP approach of non-rigid registration techniques. Namely, we assume that the closest point $\hat{q}_j$ to each $p_j$ and sign $s(p_j) = s_j^*$ 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 $F$:

$$\frac{\partial \Phi}{\partial t} \approx -\sum_{i \in N(F)} \sum_{j=1}^{h} w_j s_j^* \nabla_{F_j} \| p_j - \hat{q}_j^* \|, \quad (4)$$

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

$$\frac{\partial \Phi}{\partial t} \approx -\sum_{i \in N(F)} \sum_{j=1}^{h} w_j s_j^* (\nabla_{F_j} p_j)^T \nabla_{F_j} \| p_j - \hat{q}_j^* \| = -\sum_{i \in N(F)} \sum_{j=1}^{h} w_j \lambda_j \langle \mathbf{g}_i \rangle, \quad (5)$$

where $\langle \mathbf{g}_i \rangle = \begin{cases} s_j^* \frac{p_j - \hat{q}_j^*}{\|p_j - \hat{q}_j^*\|} & \text{if } \|p_j - \hat{q}_j^*\| > \epsilon, \\ \mathbf{0} & \text{otherwise,} \end{cases}$

where $\lambda_j$ is the barycentric coordinate of $p_j$, corresponding to $\hat{F}$ and $\mathbf{n}(q_j^*)$ is the unit normal at $q_j^*$. For $q_j^*$ 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 $s_j^*$ and closest points $q_j^*$ for all quadrature points. We terminate if all signs are negative and no intersection exists between $F$ and $\hat{C}$.

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.
We also experimented with modifications to the surface distance. After the flow step, 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 $L^\infty$ 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 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, $\tilde{F}$ is fully inside $\tilde{C}$. We now restore the fine mesh to its original vertex positions $F$, detecting and resolving collisions with the coarse mesh along the way (see Figure 8).

Jumping directly from $\tilde{F}$ to $F$ 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 $\tilde{F}$ 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:

$$U_F(t) = \frac{\tilde{F}(t + \Delta t) - \tilde{F}(t)}{\Delta t}, \quad (7)$$

where $U_F(t) \in \mathbb{R}^{np \times 3}$ are instantaneous per-vertex velocities. The positions of the coarse mesh $C(t + \Delta t)$—and in turn its similarly defined velocities $U_C(t)$—are not fixed. In general, there are an infinite number of feasible choices of $U_C(t)$ so that the repositioned coarse mesh $C(t + \Delta t)$ remains free of intersections with itself and with the re-inflating fine mesh $\tilde{F}(t + \Delta t)$. To regularize this problem, we introduce a generic energy $E(F, \tilde{C}, C)$ measuring the quality of the coarse mesh positions. We optimize this energy to update $C$ at each reverse time step:

$$\min_{C(t + \Delta t)} E \left( F, \tilde{C}, C(t + \Delta t) \right) \quad \text{subject to:} \quad (8)$$

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

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

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 $F(t)$, coarse mesh $C(t)$ and desired velocities $U_F(t)$ and $U_C(t)$, where $U_F(t)$ is a descent direction minimizing $E$. The black box outputs new adjusted velocities $U_F^\prime(t)$ and $U_C^\prime(t)$ satisfying the non-intersection constraints 9-10.

![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.](image-url)
The simplest energies are those measuring vertex displacement in a physically-based simulation parlance, this is tantamount to assigning least squares sense:

\[ E_{\text{vol}} = \int_{V_{\Omega}(C(t + \Delta t))} 1 \, dV = \int_{C(t + \Delta t)} x \cdot n \, dA, \quad (13) \]

\[ \left\{ \begin{array}{c}
\nabla E_{\text{vol}}(t) \sim -\nabla E_{\text{vol}} = -N(t + \Delta t), \\
\end{array} \right. \]

where \( N \) are the area-weighted vertex normals.

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

The null space of \( E_{\text{vol}} \) 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):

\[ E_{\text{varap}} = \sum_{i=1}^{n_C} \arg\min_{R \in SO(3)} \sum_{j \in N_i} \| \mathbf{e}_{ij}(t + \Delta t) - \hat{R} \tilde{\mathbf{e}}_{ij} \|^2 \quad (14) \]

\[ \left\{ \begin{array}{c}
\nabla E_{\text{varap}}(t) \sim -\nabla E_{\text{varap}} \\
\end{array} \right. \]

\[ E_{\text{varap}} = \sum_{T \in \mathcal{T}} \arg\min_{R \in SO(3)} \sum_{(ij) \in T} \| \mathbf{e}_{ij}(t + \Delta t) - \hat{R} \tilde{\mathbf{e}}_{ij} \|^2 \quad (15) \]

\[ \left\{ \begin{array}{c}
\nabla E_{\text{varap}}(t) \sim -\nabla E_{\text{varap}} \\
\end{array} \right. \]

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 \( \hat{C} \) and \( \mathcal{T} \) is a list of tetrahedra tessellating \( \hat{C} \). We use TETGEN to create a graded mesh with few auxiliary variables at internal Steiner vertices [Si 2003]. Computing gradients \( \nabla E_{\text{varap}} \) and \( \nabla E_{\text{varap}} \) involves first optimizing for “best fit rotations” \( \hat{R} \) 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 \( C(t + \Delta t) \) converges or until bisecting the step length \( \Delta t \) 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
Table 1: We report the average time per cage to flow \( t_{\text{flow}} \), and to re-inflate \( t_{\text{re}} \).

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

A 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 \( \bar{M}_1, \ldots, \bar{M}_k \). 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.
where local metrics vary between solves. In this case, the internal tet-meshing might need to be recomputed (seconds for \textit{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 \textit{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 \textit{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 & Barbić 2014]. In Figure 19, we compare to extracting the
Simulation on our coarse cage, with embedded artifact-ridden input

Solution Relative residual error

\begin{tabular}{|c|c|c|}
\hline
number of v-cycles & naive & ours \hline
10 & 10^{-15} & 10^{-15} \hline
5 & 10^{-10} & 10^{-10} \hline
10 & 10^{-5} & 10^{-5} \hline
15 & 10^0 & 10^0 \hline
20 & 10^{10} & 10^{10} \hline
15 & 10^{15} & 10^{15} \hline
20 & 10^{20} & 10^{20} \hline
\end{tabular}

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 achievesnesting 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.

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.
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. [Aksoyulu 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 Panazzolo, 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 DGG 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


There exist polyhedra that cannot be continuously flowed inside themselves [Epstein 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.

Appendix A: Flow counterexample

![Flow counterexample](image_url)

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.

References


**Algorithm 2: Shrink(\(\hat{C}, F\)) \rightarrow H**

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

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

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

**Algorithm 3: Reinflate(\(H, \hat{C}, \text{Energy}\)) \rightarrow C**

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

**Outputs:**
- \(C\) Final nested cage

**begin**
\[F_0 \leftarrow H\).first \quad \text{// Initial fine mesh}\]
\[F \leftarrow H\).pop \quad \text{// Shrunken fine mesh}\]
\[C \leftarrow \hat{C} \quad \text{// Initialize output cage}\]
While \(H\) is not empty do
\[\beta \leftarrow \beta_{\text{min}} (= 1e-2) \quad \text{// Initial step size}\]
\[E_{\text{min}} \leftarrow \infty \quad \text{// Initial minimum energy}\]
\[U_F \leftarrow H\).pop - F \quad \text{// Desired velocities for fine mesh}\]
Repeat
\[U_C \leftarrow -\beta \nabla \text{Energy}(F_0, C, \hat{C}) \quad \text{// ~ coarse mesh}\]
\[\text{/* Filter velocities, e.g. modified } [\text{Brochu \\& Bridson 2009}] \text{ or } [\text{Ainsley et al. 2012}] */\]
\[U_C \leftarrow \text{Filter}(F, U_F, C, U_C)\]
If \(\text{Energy}(C + U_C) > E_{\text{min}}\) then
\[\beta \leftarrow \beta/2 \quad \text{// Decrease step size}\]
If \(\beta < \beta_{\text{min}} (= 1e-3)\) then break // No progress
Else
\[C \leftarrow C + U_C \quad \text{// Step coarse mesh}\]
\[E_{\text{min}} \leftarrow \text{Energy}(C) \quad \text{// Update energy}\]
\[\beta \leftarrow 1.1\beta \quad \text{// Slightly increase step size}\]
If \(|U_C| < \Delta C (= 1e-5)\) then break // "Converged"
\[F \leftarrow F + U_F \quad \text{// Step fine mesh}\]
**end**