Fast Cutting of Cage Based Deformation with Walk on Spheres

HOSSAM SAEED, McGill University, Canada
MICHAEL LAMBIRI, McGill University, Canada
TESEO SCHNEIDER, University of Victoria, Canada
DEREK NOWROUZEZAHRAI, McGill University, Canada
PAUL G. KRY, McGill University, Canada

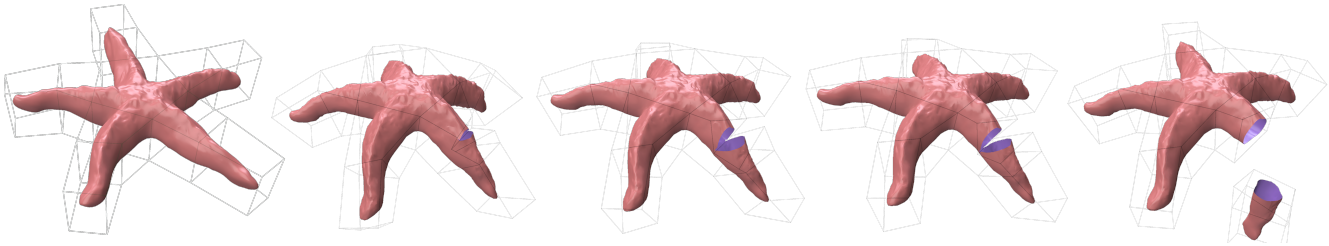

Fig. 1. Cutting this physically simulated starfish is fast because we use walk on spheres to compute the harmonic weights of the embedded mesh with respect to the cage. Weight updates during progressive cutting are an order of magnitude faster than recomputation as most walks can be reused.

While cages provide a powerful reduction model to simulate elastic deformations, interactive real-time cutting of cages has been a longstanding challenge. Traditional solvers require costly volumetric re-meshing and global matrix updates after each modification. For a mesh-free harmonic weight computation using walk on spheres (WoS), we make a key observation: only a small fraction of walks (5–10%) is invalidated by the local cut in most practical cases. We propose a novel reuse pipeline to efficiently update only weights subject to a topological modification. Our approach leverages walk on spheres as an output sensitive and mesh-free Monte Carlo method for solving Laplacian equations, to enable localized updates of harmonic weights. By identifying and reusing unaffected walks, we minimize our re-walking updates. Our method achieves speedups of 10×–40× while being statistically identical to a complete WoS weight recomputation. This enables a real time simulation of interactive cutting on complex 2D and 3D cages, and is easily generalizable to other local cage editing operations.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; *Physical simulation.*

Additional Key Words and Phrases: cage-based deformation, Monte Carlo PDE solvers, path reuse, harmonic weights, interactive cutting.

ACM Reference Format:
Hossam Saeed, Michael Lambiri, Teseo Schneider, Derek Nowrouzezahrai, and Paul G. Kry. 2026. Fast Cutting of Cage Based Deformation with Walk on Spheres. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '26), July 19–23, 2026, Los Angeles, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3799902.3811051

Authors' Contact Information: Hossam Saeed, McGill University, Montréal, Canada, hossam.mohamedsaeedmostafaahmed@mail.mcgill.ca; Michael Lambiri, McGill University, Montréal, Canada, Michael.lambiri@mail.mcgill.ca; Teseo Schneider, University of Victoria, Victoria, BC, Canada, teseo@uvic.ca; Derek Nowrouzezahrai, McGill University, Montréal, Canada, derek@cim.mcgill.ca; Paul G. Kry, McGill University, Montréal, Canada, kry@cs.mcgill.ca.

SIGGRAPH Conference Papers '26, Los Angeles, CA, USA

ACM ISBN 979-8-4007-2554-8/2026/07
https://doi.org/10.1145/3799902.3811051

1 Introduction

Cage-based simulation is a fundamental tool in computer graphics, offering an intuitive and efficient way to manipulate complex high-resolution meshes through a coarse surrounding proxy. The primary appeal of cages lies in their reduced degrees of freedom and its ability to preserve fine geometric details while providing intuitive control to animators. Furthermore, the volumetric nature of these weights facilitates easy physics-based animations, making cages an ideal bridge between low-dimensional control and high-fidelity visual output. Among various possible alternatives to propagate displacement from the change to the mesh, we focus on harmonic coordinates [Joshi et al. 2007] which allow the expression of deformations as a smooth interpolation of cage vertices.

Interactive cage editing remains a challenge because editing operations, such as cutting, introduce topological modifications to the cage. Because harmonic weights are derived from the solution of the Laplace equation over the cage's interior, traditional numerical solvers like the finite element method necessitate an explicit meshing of the interior volume or of the cage. Consequently, performing a progressive cut through the cage is computationally prohibitive; it triggers a cycle of expensive re-meshing and a full re-solve of the Laplace equations at every step. This lack of efficiency prevents true real-time "digital sculpting" or interactive design workflows, where a user might want to slice, refine, or prune a cage on the fly without the latency associated with volumetric re-meshing.

The walk on spheres algorithm [Sawhney and Crane 2020] offers a compelling mesh-free method to solve Laplace equations using sampling-based boundary evaluations rather than volumetric discretizations. This Monte Carlo solver sidesteps volumetric meshing and offers a localized output-sensitive evaluation. Since walks are completely independent, we can solve in specific parts of the domain, using the appropriate number of samples. This idea is extended by

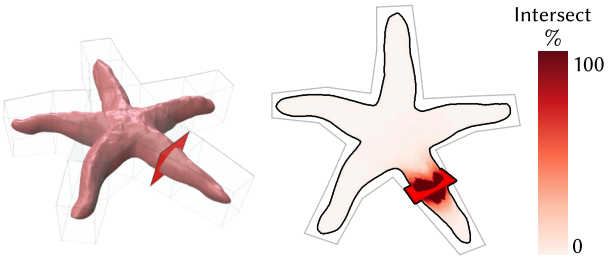

Fig. 2. Interior points far from a complete cut have only a small percentage of walks with spheres that intersect the cut. This motivates the idea of reusing walks when updating harmonic weights, with the benefits being even greater during progressive cutting.

de Goes and Desbrun [2024] who propose stochastic barycentric coordinates for use with cages, where the harmonic weight computation for interior points is formulated as Laplace equations with Dirichlet boundary conditions on the faces of the cage. The walk on spheres algorithm is employed as a fast way to compute these weights without needing to mesh the interior. Notably, doing a full re-evaluation of the walk on sphere algorithm for each cut growth step is still slow for real-time interactivity.

A key insight is that introducing a cut somewhere in the cage is usually very local in terms of effect. If a walk, prior to cutting, does not touch the cut, then there is no need to re-do it, as it is a valid sample and will play out exactly the same if replayed on the newly cut cage. In fact, in a typical cage, most walks do not intersect a local cut and remain valid for reuse. Our work utilizes this to make fast local updates, only for walks that interact with the cut. For the progressive cut in Figure 1, Figure 2 shows a visualization of the distribution of walk-cut intersections, where only **7.6%** intersect at any step. We utilize this concept to achieve an **18.9**× speedup factor for updating walk contributions compared to a full walk on spheres recomputation of weights. To handle the complexity of N-gon faces that result from arbitrary cutting in 3D, we propose a hybrid 2D-3D WoS scheme that seamlessly transitions between volumetric and surface sampling. We run a variety of tests and experiments in Section 7 to show the robustness and versatility of our pipeline.

2 Related Work

Cutting. There are several good surveys on the topic of cutting deformable models [Bruyns et al. 2002; Wang and Ma 2018; Wu et al. 2015] covering a wide range of challenges in remeshing, handling discontinuities, and managing representation complexity. Much of the literature on cutting for physically simulated deformation focuses on finite element models. Early work achieves success by restricting cuts to existing mesh boundaries [Müller and Gross 2004], while later techniques use virtual or ghost node algorithms to handle arbitrary cuts while avoiding the ill-conditioned elements and complex data-structure bookkeeping inherent in local remeshing [Molino et al. 2005; Wang et al. 2015]. Extended finite element methods address the cutting problem more accurately, where the basis functions are enriched to represent the discontinuity without needing the mesh to conform to the cut. Example works capture fine detail when cutting shells [Kaufmann et al. 2009], provide robust quadrature schemes for arbitrary cuts [Koschier et al. 2017], and wrangle multiple intersecting cuts with robust mesh Booleans [Ton-That et al. 2024]. These approaches remain distinct from our work in their reliance on high-resolution meshes. In contrast, cage-based deformations function as a form of reduced deformable model. The most relevant works in reduced models and cutting employ neural networks as discretization-free deformation models, often assisted by linear reduction [Chang et al. 2023], adapted to reduced-order elastoplasticity and fracture [Zong et al. 2023], or handling discontinuities with generalized winding numbers [Chang et al. 2025]. Unlike these neural approaches, our work leverages cage-based deformation to provide a reduced-order model that maintains explicit geometric control and supports fast topological and geometric modification thanks to our use of Monte Carlo integration.

Cage-Based Animation and Harmonic Coordinates. Among the various choices of generalized barycentric coordinates [Hormann and Sukumar 2017], harmonic coordinates have played a central role in cage-based character animation since their introduction [Joshi et al. 2007]. More recently, de Goes and Desbrun [2024] proposes a stochastic computation of these coordinates. By formulating coordinate computation as a Laplace problem with Dirichlet conditions on the cage vertices, harmonic coordinates yield smooth, well-behaved weight functions that satisfy the maximum principle and avoid spurious local extrema. However, a major limitation of barycentric coordinates is their dependence on the cage geometry. As a result, even local topology modifications introduced by cutting require recomputing the coordinate functions throughout the domain. This sensitivity to cage topology, combined with the reliance on specific surface discretizations, makes interactive topology changes to cages very difficult. Our method circumvents these limitations by leveraging walk on spheres, this formulation enables efficient updates under localized topological changes, making harmonic-coordinate-based cage editing feasible in interactive cutting scenarios.

Walk on Spheres. Originally proposed by Muller [1956] for the Dirichlet problem, the Walk on Spheres method was introduced to computer graphics by Sawhney and Crane [2020]. The method is a mesh-free approach for solving elliptic boundary value problems in volumetric domains. Since then, the method has also evolved to compute spatial derivatives [Miller et al. 2024]. Then, de Goes and Desbrun [2024] showed that using walk on spheres we can compute Monte Carlo estimates of harmonic coordinates. For a comprehensive overview of the method's variations and applications in graphics, we refer to the recent course by Sawhney et al. [2025]. Despite these advantages, all of the above works hinge on the assumption that the domain is static, thus in the event of a topology change, all walk contributions must be discarded and redone. This limitation results from the individual walks having implicit geometric validity through their distance-to-boundary constraints, and any topological modification invalidates these guarantees, leaving no mechanism in existing methods to selectively preserve correct samples. With such high recompute costs, it makes walk on spheres seem unsuitable for interactive scenarios which involve topology changes. However, while walk on spheres is traditionally treated as a global solver, its stochastic formulation inherently exhibits strong

spatial locality, since most walks interact only with nearby boundary elements and remain unaffected by distant geometric changes. In contrast, we exploit the independence and locality of walk on spheres samples to reuse previously computed contributions by identifying only the walks affected by the new geometry.

3 Background

Cage Deformation. In $\mathbb{R}^d$, a cage Ω is a set of $d-1$-dimensional faces f in $\mathbb{R}^d$, where each face has n vertices $\mathbf{v}_i \in \mathbb{R}^d$, $i = 1, \ldots, m$. In 2D, these faces are line segments, while in 3D they are planar polygons. Cages are often used as a coarse control structure to deform some internal geometry. The deformation can be defined using generalized barycentric coordinates: a point $\mathbf{p}$ inside the cage can be expressed as a linear combination of the n vertices of the cage and the barycentric coordinates. Formally,

$$\mathbf{p} = \sum_{i=1}^{n} w_i(\mathbf{p})\, \mathbf{v}_i \tag{1}$$

where $\mathbf{v}_i$ are the cage vertices, $w_i \geq 0$ and $\sum_{i=1}^{n} w_i = 1$ are the barycentric coordinates.

The weighted representation in Equation (1) provides a mechanism for space deformation. Given a set of displaced cage vertices $\mathbf{v}'_i$, the deformed position $\mathbf{p}'$ of any interior point is computed using the original weights $w_i(\mathbf{p})$ as $\mathbf{p}' = \sum_{i=1}^{n} w_i(\mathbf{p})\, \mathbf{v}'_i$.

Harmonic Coordinates. We use harmonic coordinates [Joshi et al. 2007] to propagate deformations from the cage to the interior. Each coordinate function w_i for cage vertex $\mathbf{v}_i$ is defined as the solution to the Laplace equation subject to Dirichlet boundary conditions:

$$\begin{cases} \Delta w_i(\mathbf{p}) = 0 & \text{for } \mathbf{p} \in \Omega \\ w_i(\mathbf{p}) = \phi_i(\mathbf{p}) & \text{for } \mathbf{p} \in \partial\Omega \end{cases} \tag{2}$$

where $\phi_i(\mathbf{p})$ is the piecewise linear "hat" function on the cage boundary $\partial\Omega$ such that $\phi_i(\mathbf{v}_j) = \delta_{ij}$. In traditional static contexts, Equation (2) is solved once during pre-computation. In our setting, this is unfortunately impossible as the cage geometry changes; our main challenge is efficient updates to w_i.

Walk on Spheres. Among many methods to solve the Laplace equation, the *walk on spheres* algorithm provides a mesh-free solution to Equation (2) by leveraging the *mean value property* of harmonic functions. The algorithm proceeds in steps: starting at $\mathbf{p}_0 = \mathbf{p}$, the algorithm iteratively randomly selects a point on the largest empty sphere centered at the current position $\mathbf{p}_t$, setting its radius to $r_t = d(\mathbf{p}_t, \partial\Omega)$ as the distance to the boundary. If $\mathbf{p}_t$ is close enough to the boundary (i.e., the stopping condition $r_t < \epsilon$) the walk stops at a landing point $\mathbf{x}_k$ and is assigned the PDE boundary value $\phi_i(\mathbf{x}_k)$. If not, the walk continues from $\mathbf{p}_t$ by selecting a new sphere. Finally, the barycentric coordinate $w_i(\mathbf{p})$ is estimated as the expected value of the boundary function ϕ_i over N independent paths

$$w_i(\mathbf{p}) \approx (1/N) \sum_{k=1}^{N} \phi_i(\mathbf{x}_k). \tag{3}$$

This is ideal for cage-cutting as it avoids remeshing the interior volume. The independence of individual paths enables localized, output-sensitive evaluations, a property we exploit in our method.

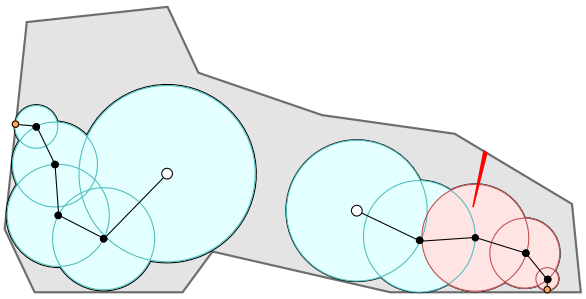

Fig. 3. Walks are classified by their interaction with the cut. Intersecting walks (right) are outdated, while others (left) are valid for reuse.

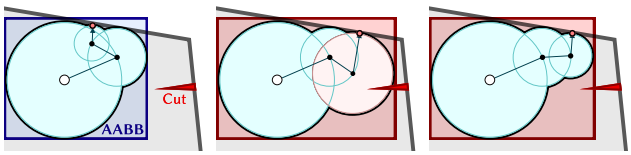

Fig. 4. AABBs provide cheap intersection tests against the cut. From left to right: a non-intersecting walk (reused), a true intersection (to re-walk), and a false positive as the conservative AABB triggers an unnecessary update.

4 Cutting Cages in 2D

We first present the essential concepts of our method in a 2D setting, before treating issues related to cutting 3D cages, presenting details on computing deformation gradients, and parallel computation of progressive cutting for physics simulation.

Our method reuses walks when the cage is modified by a cut. For a walk to be reusable, it must satisfy the minimal distance-to-boundary condition at every step, meaning that each sphere must remain contained in the interior of the cage. In the event of an intersection with the cut, this condition is violated (Figure 3).

To reuse the contribution from a specific walk, none of its spheres should intersect the newly introduced geometry. This ensures that the walk would have taken the exact same path in the modified cage. Consequently, the update process requires: (1) identifying which walks intersect the cut, (2) removing the old contributions of those intersecting walks, (3) resampling the invalidated walks without introducing bias, and (4) updating the weights for the subset of reusable walks that require only an on-boundary-segment coordinate shift.

4.1 Walk-Cut Intersection Check.

It is straightforward to check if a sphere intersects the cut geometry. However, storing every sphere of every walk is memory intensive, and running an intersection test on each one is impractical. With just 2,500 interior points, 16,000 walks per point, and a maximum of 256 steps per walk, over 10^{10} total steps would need to be stored and processed. To make this practical, we instead maintain and update a single k-DOP [Klosowski et al. 1998] per walk, enclosing all its steps, which provides a conservative intersection test with the new geometry. For simplicity, we will use axis-aligned bounding boxes (AABBs) in this section, as shown in Figure 4. The bounding volume

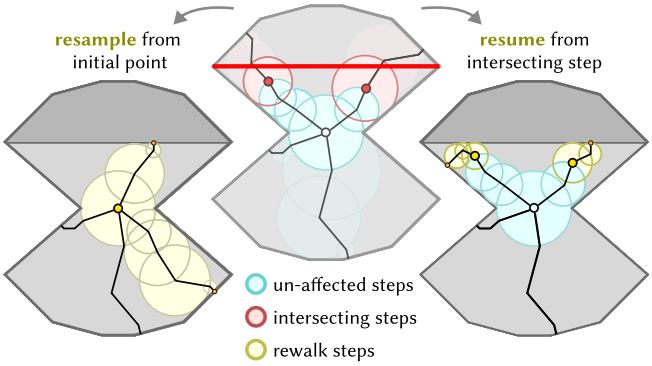

Fig. 5. **Unbiased Rewalking.** Restarting walks from the origin (left) shifts the sample distribution away from the cut. Resuming from the first point of intersection (right) maintains the correct conditional probability, producing an unbiased estimate of the harmonic weights.

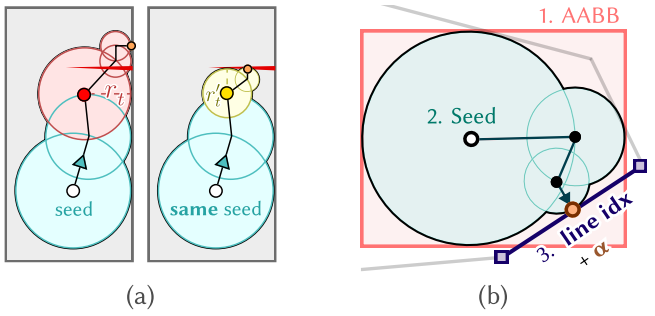

Fig. 6. (a) Replaying a walk up to the first intersecting step $\mathbf{p}_t$, and resuming it using the updated distance r'_t to the new boundary. (b) The three components cached for efficient updates: an AABB and landing data are stored per walk, alongside a single random seed per query point.

may trigger more rewalks than strictly necessary, but is used due to storage efficiency and $O(1)$ checks.

4.2 Re-Walking and Unbiased Weight Updates

When a walk's bounding volume intersects a new cut, its contribution to the cumulative weights must be updated. We first subtract the invalidated walk's contribution from the cumulative sum.

However, simply starting a new independent walk from $\mathbf{p}_0$ introduces bias. This is because we implicitly condition on *survival*: we retain old walks specifically because they *avoid* the cut, while new walks from $\mathbf{p}_0$ have no such filter. To maintain the correct estimator distribution (Fig. 5), we must resample conditional on the path entering the invalidated region. We achieve this by replaying the walk from its origin and *resuming* from the exact state $\mathbf{p}_t$ where the sphere is invalidated by the cut.

To maintain a bias-free estimate, we must resume the walk from the exact state it was in when it first encountered the cut. Although we do not store every step, we deterministically reproduce the walk by storing a single random seed per query point. We initialize the random number generator using a combination of the point's seed and the specific walk index.

This allows us to efficiently *replay* the specific walk that triggered the bounding volume intersection to find the first step t where the sphere $S(\mathbf{p}_t, r_t)$ intersects the cut. We then resume the walk from $\mathbf{p}_t$ using the updated distance to the boundary $r'_t = d(\mathbf{p}_t, \partial\Omega_{new})$, as Figure 6(a) shows. Once the resumed walk reaches the boundary, we update the point's harmonic weights and the walk's bounding volume.

4.3 Progressive Cut Weight Updates without Re-walk

Our focus is on progressive cutting. That is, the cutting process is incremental and grows only by a small amount at each time step. Thus, bounding volume intersection checks are performed against only the new part of the cut in the current step, rather than the whole edge produced by the cut. This often means fewer intersecting walks per step than a one-step cut. Nevertheless, some walks terminate on the old part of the cut edge, and the geometric change of the cut edge dictates a different Dirichlet condition at that termination point.

A walk that does not intersect the cut is reusable as is; however, its contribution to the weights may still require updating if it terminates on an edge modified by the cut. This can happen in two different ways: (1) if the walk terminates on the edge that was split by a cut (at the first step), or (2) the edge being extended by the progressive cut. In such scenarios, no re-walking is necessary. However, the weight contribution must still be updated to account for the change in the edge's parameterization. When an edge is split by a cut, the barycentric coordinate along the edge (parameterized by α) may shift for walks that land on that edge. We update the weight by replacing the old contribution (the original α) with a new contribution (the new α value in the modified edge geometry).

4.4 Cached Data

To facilitate efficient re-walking and updates, we store the three components shown in Figure 6(b): an AABB, the random seed (per point), and the landing data (line index and parameter α). The landing data is used for efficiently subtracting weights from invalid walks without performing a full re-walk.

Using standard 32-bit types, our baseline implementation with AABBs requires only 24 bytes per walk. Using k-DOPs increases this footprint linearly with k, yet total storage remains well within practical limits for maintaining millions of walks in memory.

5 Cutting Cages in 3D

The computation of harmonic weights in 3D cages is very similar to the 2D case, as is our walk reuse method for cutting. However, there are a few key differences as explained below.

5.1 Boundary Interpolation on N-Gons

Unlike 2D line segments, interpolating weights on a 3D planar face is non-trivial for arbitrary polygons. To handle general N-gon faces without complex remeshing, we employ a nested 2D walk on spheres algorithm (Figure 7). When a 3D walk lands on a polygonal face, it continues as a secondary 2D walk restricted to the plane of that face to determine the final harmonic weights at the face

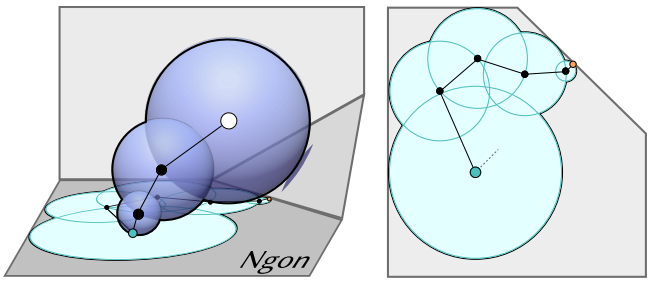

Fig. 7. A 3D walk hitting an N-gon face (left) continues in-plane as a 2D walk (right). This enables weight computation for arbitrary polygons by treating edges as the absorbing boundary.

vertices. Effectively, this treats the face itself as a 2D cage. Since the nested walk terminates on an edge, we store only two vertex IDs and their weights, maintaining a constant-size 4-vector, regardless of the number of vertices in the N-gon.

While this approach is robust for N-gons, it is computationally more expensive due to the variance inherent in Monte Carlo estimation. Therefore, for simpler primitives, we opt for analytic solutions. We use barycentric interpolation for triangles. For convex quadrilaterals, we evaluate weights analytically using mean value coordinates [Floater 2003], the unnormalized weight w_i associated with $\mathbf{v}_i$ is

$$w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|\mathbf{v}_i - \mathbf{p}\|}, \tag{4}$$

where α_i is the angle subtended by the edge $(\mathbf{v}_i, \mathbf{v}_{i+1})$ at $\mathbf{p}$. In our implementation, we prioritize these analytic methods whenever possible, reserving the nested 2D walk only for complex N-gons where analytic formulations are unavailable. The key advantage of this hybrid approach is that when a face is topologically modified by a cut, we avoid the process of re-triangulating the face altogether.

5.2 Weight Updates in 3D

We define a 3D cut via a line and a cutting direction, utilizing a half-edge data structure to manage topology updates while ensuring faces remain planar in the rest state. To efficiently update the walk data, we maintain a minimal state of the incremental cut: (1) indices of faces currently being split, (2) face pairs representing the two sides of the progressive cut, (3) a polygon representing the new geometry introduced in the current step (for intersection testing), and (4) the indices of vertices created or displaced. This information is sufficient to perform our weight updates. As summarized in Figure 8, the update logic follows the topological changes:

(1) **Intersecting Walks:** For walks where the bounding volume intersects the new cut geometry, we subtract the old contribution and resume the walk from the point of intersection (following the unbiased strategy in Section 4.2).
(2) **Modified Faces:** For walks not intersecting the cut but terminating on a face modified by it (e.g., a face split by the cut or the advancing cut tip), no re-walking is required. We simply re-evaluate the boundary coordinates (as in Section 5.1) of the landing point on the modified geometry and update the weight accumulation.

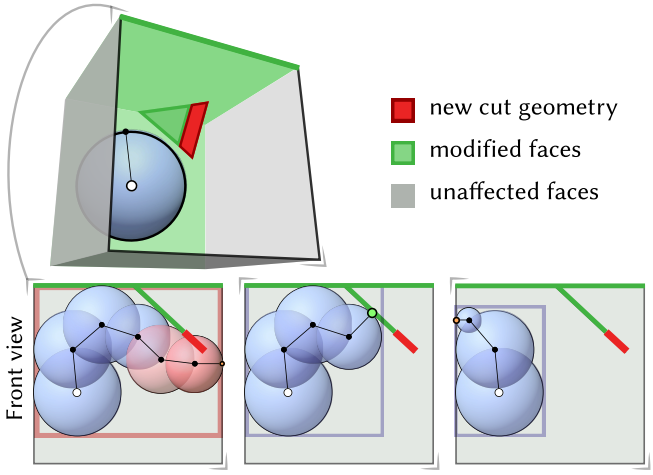

Fig. 8. **3D Weight Update Cases on Cut Progression.** The three subfigures show a 2D front view of the 3D cage, as an incremental cut introduces new cut geometry. A walk intersecting it (left) requires a full re-walk. A much simpler on-face coordinate update is required for walks landing on a face modified by the cut (middle). Otherwise, we directly reuse weight contributions from unaffected walks (right).

6 Simulation and Implementation Details

We have now explained our approach to efficiently update the harmonic weights of points inside the cage as it undergoes topological and geometric modification. With the goal of using these cage deformations as part of an elastic simulation, we also need gradients of the harmonic weights to compute deformation gradients. Below we briefly describe these details along with our approach to elastic simulation, as well as other details related to our parallel implementation, including accelerated distance queries, weight corrections, and cutting of the embedded mesh.

6.1 Elastic Simulation

We simulate the cage's elastic response by sampling the deformation gradient at quadrature points distributed throughout its volume via Poisson disc sampling [Bridson 2007]. A significant advantage of this grid-free approach is that topological changes, such as new cuts, require no special handling; quadrature points near a boundary simply terminate their walks according to the standard WoS ϵ-shell condition. Furthermore, by evaluating the deformation gradient at a distance $r > \epsilon$ from the boundary, we bypass the $r^{-1/2}$ crack-tip singularity, ensuring bounded values and numerical stability.

We compute deformation gradients using the boundary integral estimator of Sawhney and Crane [2020], which extracts the gradient directly from the primary walks. The estimator weights boundary values by the direction of the walk's first step, requiring no additional perturbed walks. Consequently, our implementation maintains a single bounding volume per walk, whether we are computing the solution, its gradient, or both.

In our elastic simulations, we use this estimated deformation gradient and the Saint Venant-Kirchhoff model. The variance in the walk-on-spheres estimates of deformation gradient, leads to a nonzero

strain when the cage is in rest pose, which is not physically correct. We address this by defining a plastic strain at each quadrature point to ensure zero strain at rest. We use a single Newton step backward Euler method for numerical integration in all examples. To maintain smoothness during cutting, newly created vertices of the initial cut inherit the interpolated position and velocity of their neighbors in the deformed mesh. For the tip of the cut in progressive cutting, we simply let the elastic simulation drive the tip vertices toward their new equilibrium positions.

6.2 Parallel Implementation and Acceleration

We implement our system using NVIDIA Warp in Python [Macklin 2022], a framework that JIT-compiles kernels into high-performance CUDA modules. Each walk or update is run on a GPU thread, providing the throughput necessary for interactive real-time updates. We maintain cage geometry in vertex and face arrays; for distance queries, we implement both an exact all faces search and a $O(1)$ uniform grid lookup which is exact for all cells but those that contain the medial axis. All of our results are reported for exact distance queries.

6.3 Embedded Mesh Cutting and Quality

To eliminate embedded mesh reconstruction errors as well as *wiggling* artifacts caused by Monte Carlo variance during interactive cutting, we compute weight corrections. In parallel, for every point $\mathbf{p}_i$, we compute the minimal affine correction $\Delta\mathbf{w}_i$ required to exactly recover the geometry of the rest position:

$$\min_{\Delta\mathbf{w}} \|\Delta\mathbf{w}\|^2 \text{ s.t. } \sum_j (w_{ij} + \Delta w_{ij})\mathbf{v}_j = \mathbf{p}_i, \ \sum_j \Delta w_{ij} = 0, \quad (5)$$

where $\mathbf{v}_j$ are the cage vertices and $\mathbf{p}_i$ is the evaluation point. This constrained least-squares problem reduces to a 4×4 linear system dependent only on the rest cage configuration; we therefore precompute the system's inverse once and apply the correction efficiently on the GPU.

As the cut moves through the interior geometry, we perform local mesh modifications on the embedded mesh by splitting faces that intersect the cutting plane. New vertices and triangles are created to permit the embedded mesh to animate correctly at the cut. These are usually low in numbers per cut growth step. Unlike other vertices in the mesh that undergo incremental weight updates, these newly created vertices require an initialization of their harmonic weights. However, by construction, they live on the new boundary introduced by the cut, and thus these walks terminate instantaneously and so the time spent on them is minimal.

7 Results

We evaluate our method's efficiency and fidelity on a workstation with an NVIDIA RTX 5090 (32GB VRAM) and 64GB of system RAM. We validate reuse correctness, analyze efficiency of bounding strategies, and demonstrate real-time performance across a variety of scenarios. Table 1 shows the cages (Figure 11) and embedded meshes used in our experiments. In Figures 16, 17, 18, and 19, we show qualitative results, demonstrating interactive cutting and deformation fidelity.

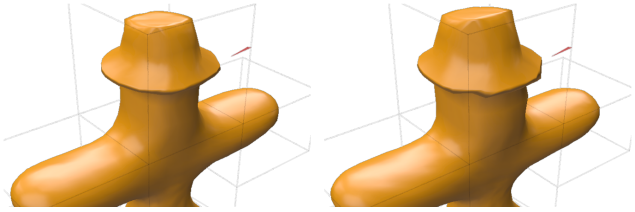

Fig. 9. Verifying the seed replay strategy. Left: Replaying the walk with the stored random number generator seed avoids bias by maintaining consistency up to the first invalid step. Right: Restarting from the original point without seed replay introduces a noticeable distortion effect.

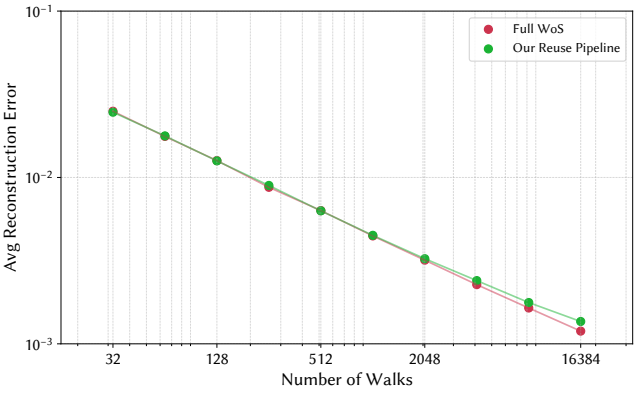

Fig. 10. Reuse pipeline convergence. Our estimator (green) achieves statistical consistency and the standard $1/\sqrt{N}$ convergence rate of the baseline WoS (red). Tested on Dude following the Figure 9 configuration; the reuse strategy introduces no significant bias.

7.1 Analysis of Walk Reuse and Intersection

We first analyze the efficacy of our reuse strategy. We focus on the number of invalidated walks as it is the main factor driving the performance gains.

Estimator Validity. We verify that our reuse logic preserves the mathematical properties of the baseline WoS estimator. Figure 9 demonstrates the visual necessity of seed replay for an unbiased reconstruction. Without replaying walks using a stored random number generator seed, the coordinates produce noticeable distortions near the cut. Figure 10 confirms that our pipeline maintains the standard $1/\sqrt{N}$ convergence rate. This ensures that our performance gains do not come at the cost of physical accuracy.

Cut Locality. We observe a strong correlation between the cut location and the percentage of invalidated walks. As shown in Figure 12, cuts performed on the extremities result in significantly fewer intersections. This confirms that the locality of the cut is effectively exploited by our algorithm, minimizing the need for re-computation.

Bounding Volume Comparison. Efficiently detecting which walks intersect the newly introduced cuts is critical. While checking exact intersections against all cut facets is infeasible, we compare the false positive (FP) rates of bounding volume hierarchies: AABBs

Table 1. **Performance Statistics.** Our method achieves 10× to 40× speedups on practical cases (Figure 11). Experiments use 2048 walks/vertex with ≈ 3% cut increments. Ours (ms) is the weight update time, comprising bounding volume intersection checks (BV Check), re-walking intersected paths (Re-walk), and updating on-face landings (On-Face). Naive (ms) denotes full recomputation; Total (ms) is the end-to-end frame time including coordinate/gradient updates, elastodynamics solve, and system overhead. Memory is the GPU walk cache footprint. All reported times are averages over the cut steps.

	Model	F_{cage}	V_{cage}	V_{mesh}	Re-walk (%)	On-Face (%)	BV Check Time (ms)	Re-walk Time (ms)	On-Face Time (ms)	Time (ms) (Ours)	Time (ms) (Naive)	Speed-Up	Memory (GB)	Total (ms)
3D	Dude	35	34	2314	4.81	6.05	1.18	7.13	0.41	**9.55**	112.27	**11.8**×	0.55	20.1
3D	Starfish	53	50	8158	3.22	5.58	3.54	20.66	1.18	**27.17**	513.77	**18.9**×	1.93	40.2
3D	Key	81	74	1256	2.68	4.99	0.69	3.84	0.28	**5.65**	86.16	**15.2**×	0.30	27.0
3D	Octopus	407	375	2784	0.36	0.44	1.48	15.06	0.29	**20.22**	845.00	**41.8**×	0.66	198.4
Adv.	Jelly	21	12	2210	14.87	5.54	1.14	12.18	0.67	**14.86**	62.36	**4.2**×	0.52	23.6
Adv.	Sphere	12	8	10 242	15.87	28.53	4.71	36.08	4.44	**46.65**	130.89	**2.8**×	2.42	52.1

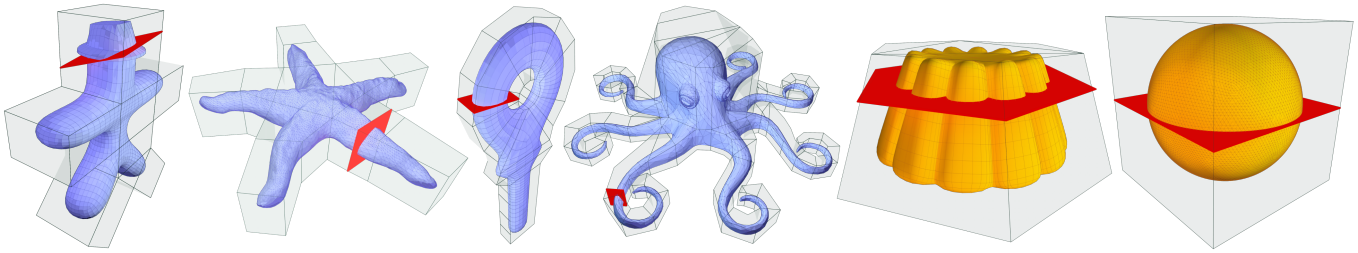

Fig. 11. Showing the cages, embedded meshes, and cuts for our experiments: Dude, Starfish, Key, Octopus, Jelly, and Sphere. Adversarial cases are in yellow.

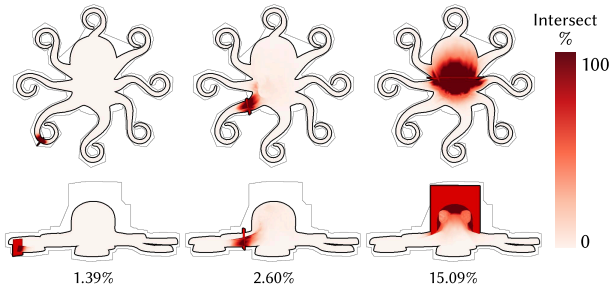

Fig. 12. Visualizing walk-cut intersection density on Octopus. Notice the sparsity of high-intersection zones in the cases of localized cuts (left & middle) as opposed to a cut going through the topological center (right).

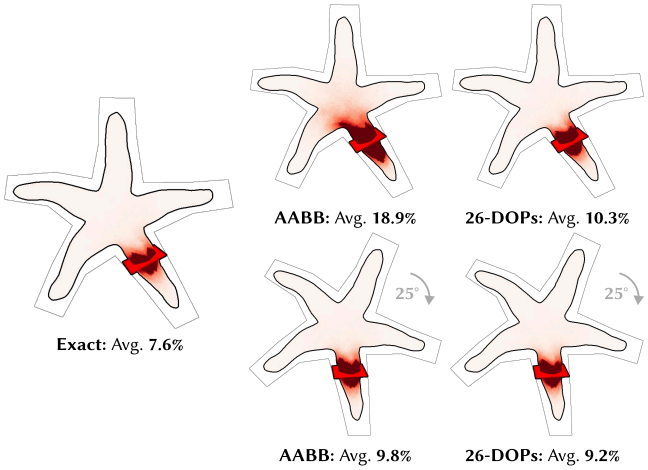

Fig. 13. Intersection density comparison on Starfish. K-DOPs provide significantly fewer false positives than AABBs, which are substantially more sensitive to the relative orientation of the cut.

and k-DOPs. Figure 13 shows that k-DOPs significantly outperform AABBs. By maintaining tight bounds independent of orientation, k-DOPs avoid the high false positive rates that AABBs exhibit on non-axis-aligned cuts. We note that while 1-Sphere bounds are memory-efficient and orientation-invariant, they yield approximately 2× more false positives than AABBs. Consequently, we default to k-DOPs to strike the best balance between memory overhead and culling efficiency.

7.2 Performance

We measure the computational speedup of our incremental approach against a baseline that re-runs the full walk on spheres solve at every time step. A summary of performance statistics across various scenarios (Figure 11) is provided in Table 1.

Speedups and Re-Walk Dominance. Our method achieves speedups ranging from 10× to 40× for typical cutting scenarios. The data reveals that the total update time is dominated by the *Re-Walk* time, which is driven by intersection counts. As supported by Figure 14, update time scales linearly with the number of intersected walks.

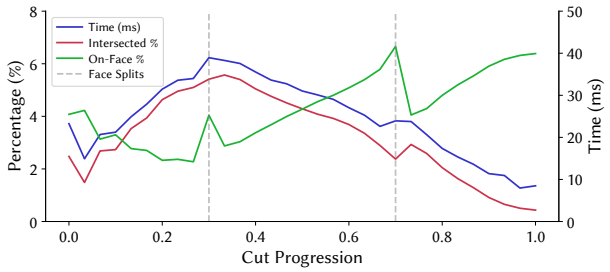

Fig. 14. Evolution of update statistics and timing during a progressive cut on the Starfish mesh. Update time tracks the intersection percentage, which peaks early and then decays as the expanding cut obstructs longer re-walk paths. Conversely, the proportion of *on-face* updates rises as the cut grows, with spikes occurring during face splits.

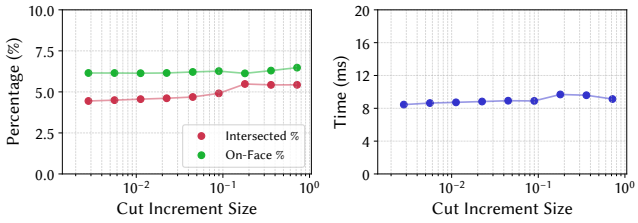

Fig. 15. Average update stats and performance per step vs.cut increment size (normalized by complete cut length) on Dude. Performance remains stable even for large increments, showing minimal dependency on step size.

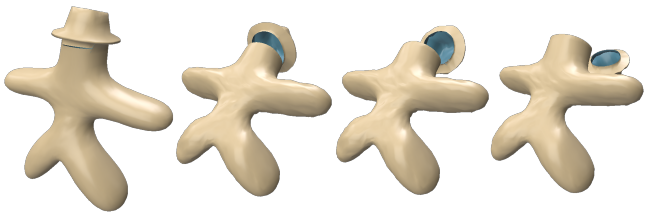

Fig. 16. We show the Dude model losing his hat to the wind.

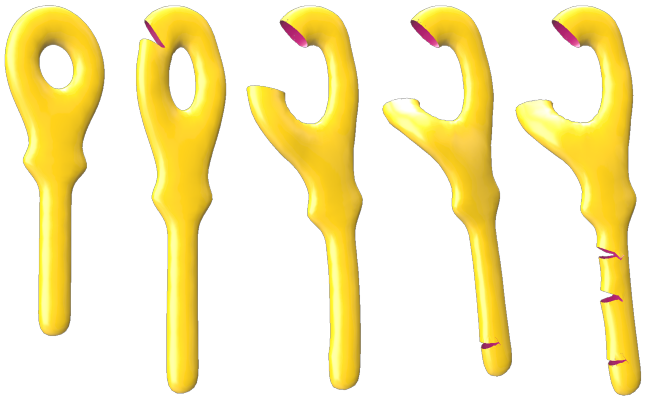

Fig. 17. Multiple progressive cuts on Key provide interesting dynamics during physics simulation.

Memory and Scalability. Memory is dominated by the walk cache reported in Table 1, which consists of k-DOP bounding volumes (104B/walk for k=26) and landing data (20B/walk). While 26-DOPs require 124B per walk, which totals 0.3–2.4GB for our models, this requirement scales linearly with evaluation points and walks, remaining *independent* of cage complexity. This represents a tunable trade-off: switching to standard AABBs reduces memory to 44B per walk (≈ 65% reduction) for a minor 1.4× performance penalty on average.

Total Cost. The total step time (Table 1) encompasses coordinate, gradient, and physical simulation updates. For Starfish, a 40 ms step comprises 27 ms for coordinate updates and 2 ms for gradient updates; the latter scales with the number of quadrature points (600 for Starfish). The remaining 11 ms is explicitly distributed across physics and topological maintenance: 4 ms for stiffness matrix assembly (on GPU), 2 ms for the sparse linear system solve (on CPU), and 5 ms for remaining updates including topology, state, and local inner-mesh retriangulation. In contrast, for Octopus, the simulation overhead scales heavily with the large number of cage vertices (375) and quadrature points (1200). This results in a 198 ms total step time, which specifically includes 33 ms for stiffness assembly and 71 ms for the sparse solve. We note that under reasonable cage complexities, step time is dominated by the coordinate updates rather than the elastodynamics solve.

On-Face Updates. While a large portion of walks may require *on-face* updates (for walks that landed on modified faces), the computational cost for these on-face updates is negligible. This validates our strategy of separating these quick updates from full rewalks.

Adversarial Cases. To stress-test our method, we subjected it to scenarios where the cut is global and the domain is convex, such as a sphere enclosed in a box or a jelly object in a hexagonal prism. Even in these high-intersection cases where locality is minimized, we maintain speedups of 3× to 4× over the baseline per step.

Parameters and Dimensionality. Our performance is largely independent of the cut increment size, as illustrated in Figure 15. This suggests our method is robust to reasonable speeds of the cutting tool. Analyzing the pipeline in 2D, the locality behavior and our reuse strategy remain effective, we observe that the benefit is more pronounced in 3D. This is expected, as walks in 3D exhibit greater sparsity and a lower probability of intersecting a local surface cut compared to 2D.

8 Discussion and Conclusion

We have presented a pipeline for the fast update of cage-based coordinates, leveraging an efficient reuse strategy for walk-on-spheres paths. Unlike previous approaches, our method isolates geometric modifications to validate only the affected subset of paths, achieving speedups of 10×–40× in practical scenarios. Crucially, this efficiency

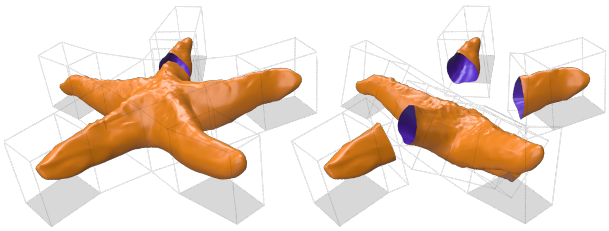

Fig. 18. Performing multiple cuts on Starfish. Our weight updates can handle arbitrarily many connected components. Grey shaded cage faces are constrained not to move.

comes without introducing numerical bias or increasing the variance of the Monte Carlo estimator, preserving the exactness of the original formulation while enabling interactive, real-time cutting.

Limitations. Despite these advantages, our method has limitations. First, performance gains rely on the locality of the cut; in highly convex shapes or during global slicing operations where the cut intersects a majority of walks, the speedup diminishes closer to the baseline. Second, we strictly require cage faces to be planar. This ensures a well-defined termination for 3D walks and allows subsequent on-face walks to remain valid within the 2D polygon.

Future Work. There are several promising directions for extension. Our reuse logic extends naturally to interactive cage editing: obstructing operations (e.g., cuts) can trigger a *re-walk*, while extending operations (e.g., extrusions) can simply *resume* walks from former landing points, enabling instantaneous feedback. Additionally, by treating the original coordinates as a control variate with weight correction, it may be possible to simulate cutting on high-resolution meshes using very low walk counts, reserving higher sample counts only for gradient estimation. Performance could be further optimized by integrating state-of-the-art distance acceleration structures to scale better with cage resolution, and by exploring tighter bounding volumes for intersection checks.

Acknowledgments

The Bellairs Workshop on Computer Animation was instrumental in the conception of the research presented in this paper. This work was also partially supported by NSERC grants DGECR-2021-00461, RGPIN-2021-03707, RGPIN-2024-04523, and RGPIN-2025-04951, as well as a gift from Adobe Research. We thank Ian Christopher ("Starfish", CC BY 4.0) and DiabaseEngineer ("Flexible Octopus", CC BY-SA 4.0) for their 3D models.

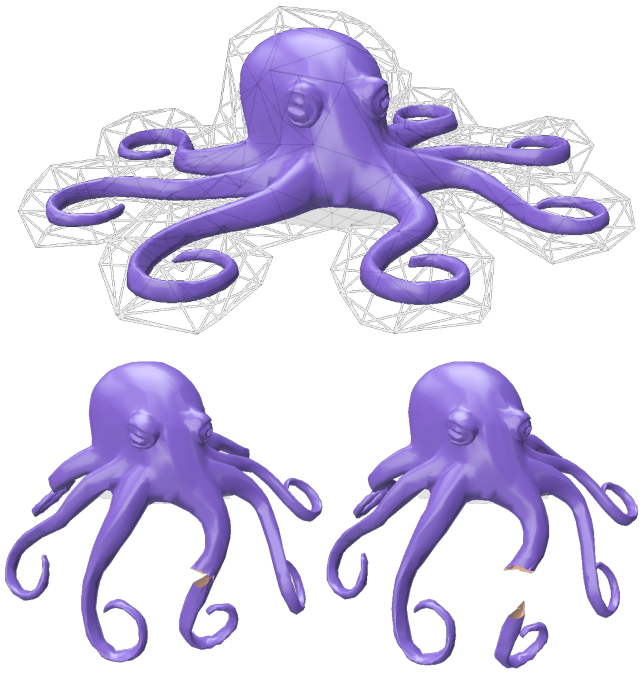

Fig. 19. Partial and complete cuts of Octopus. Here the cage is computed by alpha wrapping [Portaneri et al. 2022] a simplified mesh[Garland and Heckbert 1997], which makes the cage less artist friendly than our other examples.

References

R. Bridson. 2007. Fast Poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches (SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 22–es. https://doi.org/10.1145/1278780.1278807

C. D. Bruyns, S. Senger, A. Menon, K. Montgomery, S. Wildermuth, and R. Boyle. 2002. A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools. *The Journal of Visualization and Computer Animation* 13, 1 (2002), 21–42. https://doi.org/10.1002/vis.275

Y. Chang, P. Y. Chen, Z. Wang, M. M. Chiaramonte, K. Carlberg, and E. Grinspun. 2023. LiCROM: Linear-Subspace Continuous Reduced Order Modeling with Neural Fields. In *SIGGRAPH Asia 2023 Conference Papers (SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 111, 12 pages. https://doi.org/10.1145/3610548.3618158

Y. Chang, M. Liu, Z. Wang, P. Y. Chen, and E. Grinspun. 2025. Lifting the Winding Number: Precise Discontinuities in Neural Fields for Physics Simulation. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25)*. Association for Computing Machinery, New York, NY, USA, Article 25, 11 pages. https://doi.org/10.1145/3721238.3730597

F. de Goes and M. Desbrun. 2024. Stochastic Computation of Barycentric Coordinates. *ACM Trans. Graph.* 43, 4, Article 42 (July 2024), 13 pages. https://doi.org/10.1145/3658131

M. S. Floater. 2003. Mean value coordinates. *Comput. Aided Geom. Des.* 20, 1 (March 2003), 19–27.

M. Garland and P. S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 209–216. https://doi.org/10.1145/258734.258849

K. Hormann and N. Sukumar (Eds.). 2017. *Generalized Barycentric Coordinates in Computer Graphics and Computational Mechanics*. Taylor & Francis, CRC Press, Boca Raton.

P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph.* 26, 3 (July 2007), 71–es. https://doi.org/10.1145/1276377.1276466

P. Kaufmann, S. Martin, M. Botsch, E. Grinspun, and M. Gross. 2009. Enrichment textures for detailed cutting of shells. *ACM Trans. Graph.* 28, 3, Article 50 (July 2009), 10 pages. https://doi.org/10.1145/1531326.1531356

J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. 1998. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 4, 1 (1998), 21–36.

D. Koschier, J. Bender, and N. Thuerey. 2017. Robust eXtended finite elements for complex cutting of deformables. *ACM Trans. Graph.* 36, 4, Article 55 (July 2017), 13 pages. https://doi.org/10.1145/3072959.3073666

M. Macklin. 2022. Warp: A High-performance Python Framework for GPU Simulation and Graphics. https://github.com/nvidia/warp. NVIDIA GPU Technology Conference (GTC).

B. Miller, R. Sawhney, K. Crane, and I. Gkioulekas. 2024. Differential Walk on Spheres. *ACM Trans. Graph.* 43, 6, Article 174 (Nov. 2024), 18 pages. https://doi.org/10.1145/3687913

N. Molino, Z. Bao, and R. Fedkiw. 2005. A virtual node algorithm for changing mesh topology during simulation. In *ACM SIGGRAPH 2005 Courses (SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 4–es. https://doi.org/10.1145/1198555.1198574

M. Müller and M. Gross. 2004. Interactive virtual materials. In *Proceedings of Graphics Interface 2004 (GI '04)*. Canadian Human-Computer Communications Society, Waterloo, CAN, 239–246.

M. E. Muller. 1956. Some Continuous Monte Carlo Methods for the Dirichlet Problem. *The Annals of Mathematical Statistics* 27, 3 (1956), 569–589. http://www.jstor.org/stable/2237369

C. Portaneri, M. Rouxel-Labbé, M. Hemmer, D. Cohen-Steiner, and P. Alliez. 2022. Alpha wrapping with an offset. *ACM Trans. Graph.* 41, 4, Article 127 (July 2022), 22 pages. https://doi.org/10.1145/3528223.3530152

R. Sawhney and K. Crane. 2020. Monte Carlo geometry processing: a grid-free approach to PDE-based methods on volumetric domains. *ACM Trans. Graph.* 39, 4, Article 123 (Aug. 2020), 18 pages. https://doi.org/10.1145/3386569.3392374

R. Sawhney, B. Miller, I. Gkioulekas, and K. Crane. 2025. State of the Art in Grid-Free Monte Carlo Methods for Partial Differential Equations. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Courses (SIGGRAPH Courses '25)*. Association for Computing Machinery, New York, NY, USA, Article 3, 8 pages. https://doi.org/10.1145/3721241.3734001

Q. M. Ton-That, P. G. Kry, and S. Andrews. 2024. Generalized eXtended Finite Element Method for Deformable Cutting via Boolean Operations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '24)*. Eurographics Association, Goslar, DEU, 1–13. https://doi.org/10.1111/cgf.15184

M. Wang and Y. Ma. 2018. A review of virtual cutting methods and technology in deformable objects. *The International Journal of Medical Robotics and Computer Assisted Surgery* 14, 5 (2018), e1923. https://doi.org/10.1002/rcs.1923 e1923 RCS-17-0190.R1.

Y. Wang, C. Jiang, C. Schroeder, and J. Teran. 2015. An adaptive virtual node algorithm with robust mesh cutting. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '14)*. Eurographics Association, Goslar, DEU, 77–85.

J. Wu, R. Westermann, and C. Dick. 2015. A Survey of Physically Based Simulation of Cuts in Deformable Bodies. *Computer Graphics Forum* 34, 6 (2015), 161–187. https://doi.org/10.1111/cgf.12528

Z. Zong, X. Li, M. Li, M. M. Chiaramonte, W. Matusik, E. Grinspun, K. Carlberg, C. Jiang, and P. Y. Chen. 2023. Neural Stress Fields for Reduced-order Elastoplasticity and Fracture. In *SIGGRAPH Asia 2023 Conference Papers (SA '23)*. ACM, 1–11. https://doi.org/10.1145/3610548.3618207