Lifted Surfacing of Generalized Sweep Volumes

YIWEN JU, Washington University in St. Louis, USA
QINGNAN ZHOU, Adobe Research, USA
XINGYI DU, Tencent, USA
NATHAN CARR, Adobe Research, USA
TAO JU, Washington University in St. Louis, USA

Computing the boundary surface of the 3D volume swept by a rigid or deforming solid remains a challenging problem in geometric modeling. Existing approaches are often limited to sweeping rigid shapes, cannot guarantee a watertight surface, or struggle with modeling the intricate geometric features (e.g., sharp creases and narrow gaps) and topological features (e.g., interior voids). We make the observation that the sweep boundary is a subset of the projection of the intersection of two implicit surfaces in a higher dimension, and we derive a characterization of the subset using winding numbers. These insights lead to a general algorithm for any sweep represented as a smooth time-varying implicit function satisfying a genericity assumption, and it produces a watertight and intersection-free surface that better approximates the geometric and topological features than existing methods.

CCS Concepts: • **Computing methodologies** → **Volumetric models**; **Mesh models**.

Additional Key Words and Phrases: Sweep volume, implicit surfaces, arrangement, winding numbers

ACM Reference Format:
Yiwen Ju, Qingnan Zhou, Xingyi Du, Nathan Carr, and Tao Ju. 2025. Lifted Surfacing of Generalized Sweep Volumes. *ACM Trans. Graph.* 44, 6, Article 248 (December 2025), 17 pages. https://doi.org/10.1145/3763360

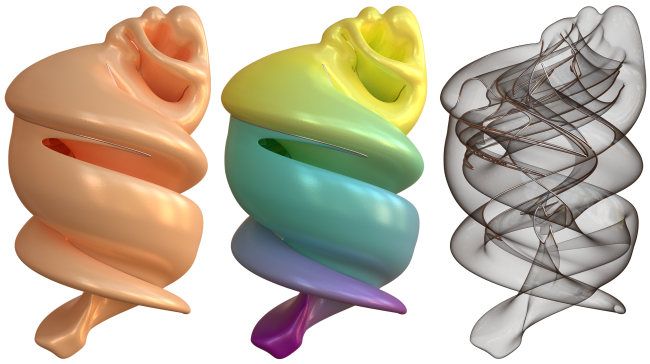


Fig. 1. The sweep boundary computed by our method for sweeping an implicit surface (the Fertility) along a down spiral while offsetting the surface inward at the same time, changing its shape and topology. Showing the surface with plain color (left), colored by time (middle), and in transparency (right). Our method produces a watertight, intersection-free surface equipped with sharp creases (metal wires).

1 Introduction

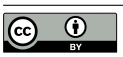
A *sweep volume* refers to the space swept by a moving, possibly transforming and deforming, solid shape (called a *brush*) over time. In solid modeling, sweeping is a classical and intuitive design metaphor, and intricate shapes can be created by combinations of brush shapes and sweep motions (see Figure 1). Other applications of sweep volumes include simulating machine milling results, computing workspace in robotics, and performing continuous collision detection in motion planning [Abdel-Malek et al. 2006].

Modeling the boundary of a sweep volume has proven to be a daunting task. As the sweep boundary is characterized by the solution of non-linear equations [Martin and Stephenson 1990], exact computation is infeasible except for simple brush shapes and sweep motions. Even approximating the sweep boundary is challenging due to its unique geometric and topological features. First, the sweep boundary may not be a smooth surface, since sharp creases can appear where instances of the brush at different times collide with each other (e.g., metal wires in Figure 1 and green curves in Figure 2 (b)). Second, different parts of the sweep boundary may become very close to each other, creating narrow gaps (see Figure 6 (b)). Lastly, the sweep boundary may consist of multiple connected components, some of which enclose "voids" that are not connected to the outside (see Figure 2 (b)). While the geometric features are important for shape design, detecting voids is essential for collision detection and workspace computation.

Despite decades of research, it remains challenging to approximate these topological and geometric features for arbitrary brush shapes and deformations while ensuring a watertight and intersection-free surface (see review in Section 3.1). While *volumetric* methods can be applied to a broad range of sweeps and produce a closed surface, the use of a grid fundamentally limits their ability to model fine geometric details, resulting in rounded creases (Figure 2 (c,d)) and artifacts at narrow gaps (Figure 6 (c,d)). While such details are better approximated by *mesh-based* methods, such methods are currently limited to simple (e.g., affine) sweep motions, may produce gaps and self-intersections, and omit the interior voids.

In this paper, we present a new method for surfacing sweep boundaries in 3D that improves existing methods in generality, robustness, and feature awareness. Our method works on any *generalized* sweep that is represented by a smooth 4D implicit function. The representation accommodates a wide range of deformations during sweeping, including those with topological changes (Figure 1). Our method better approximates the interior voids, sharp creases

Authors' Contact Information: Yiwen Ju, Washington University in St. Louis, USA, yiwen.ju@wustl.edu; Qingnan Zhou, Adobe Research, USA, qzhou@adobe.com; Xingyi Du, Tencent, USA, xingyidu@global.tencent.com; Nathan Carr, Adobe Research, USA, ncarr@adobe.com; Tao Ju, Washington University in St. Louis, USA, taoju@wustl.edu.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 1557-7368/2025/12-ART248
https://doi.org/10.1145/3763360

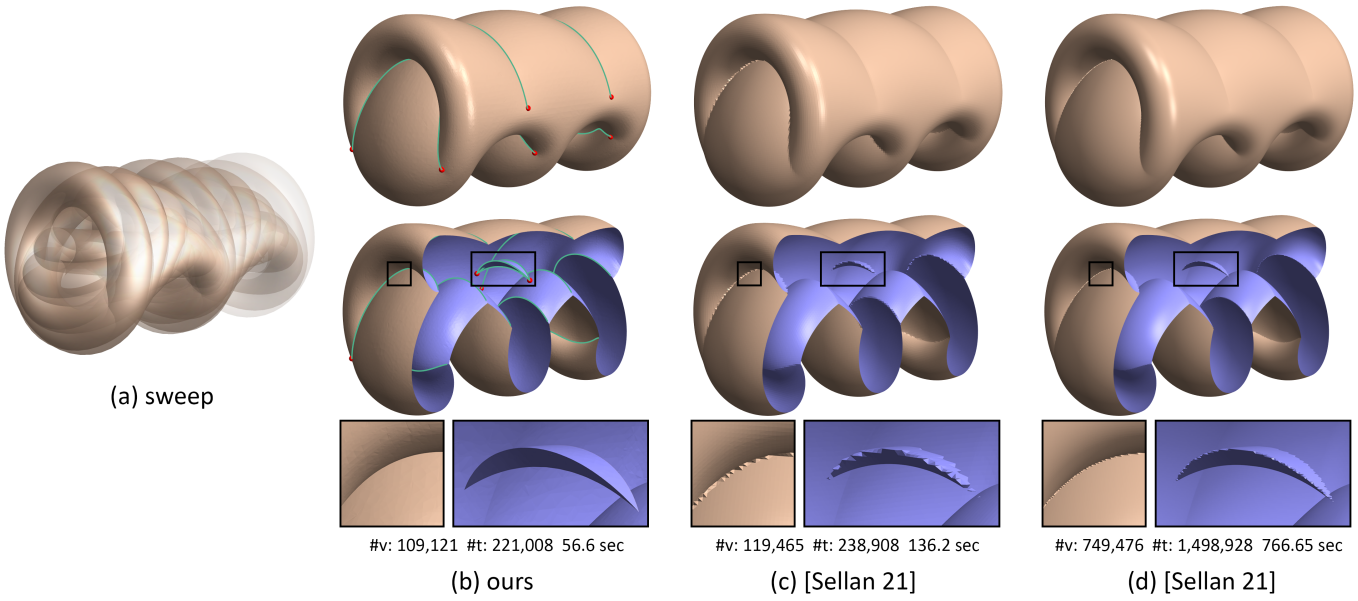


Fig. 2. Comparing sweep boundaries computed by our method (b) and the volumetric method of [Sellán et al. 2021] at grid cell size 0.005 (c) and 0.002 (d) on a moving and flipping torus (a), showing the exterior surfaces (top) and cut-away views (middle) with zoom-ins on a sharp crease and a void (bottom). While [Sellán et al. 2021] produces rounded creases, even on a fine grid, our method keeps the creases sharp.

and narrow spaces than existing methods while ensuring a closed and intersection-free surface (see Figures 2 (b) and 6 (b)).

Our method combines elements from both volumetric and mesh-based methods. Like many mesh-based methods, our method computes and trims a superset of the sweep boundary, known as the *envelope* [Pottmann and Peternell 2000; Wang and Wang 1986]. To make the process robust and general, we make two key observations (Section 4). First, an envelope in $\mathbb{R}^3$ is the projection of the intersection of two implicit hypersurfaces in $\mathbb{R}^4$, which we call the *lifted envelope*. This observation enables robust computation of the envelope via iso-surfacing. Second, the subset of the envelope that lies on the sweep boundary can be simply characterized as those bounding regions with 0 winding number, if the envelope is properly oriented. Leveraging existing packages for mesh arrangement and winding numbers [Zhou et al. 2016], this observation makes the trim step simple to implement and numerically robust.

The core of our method is a grid-based algorithm that computes the lifted envelope, a 2-manifold at the intersection of two level sets in 4D (Section 6). Our tailored algorithm improves the triangle quality of existing methods for discretizing vector-valued level sets in high dimensions (see review in Section 3.2), and it ensures a correct orientation needed for trimming. Coupled with adaptive grid refinement (Section 7), our algorithm can be applied to sweep boundaries with complex topology and geometry.

To summarize, our work makes the following theoretical and algorithmic contributions:

(1) A novel characterization of the sweep boundary, based on level set intersection and winding numbers, that applies to generalized sweeps in any dimension.
(2) An algorithm for computing sweep boundaries in $\mathbb{R}^3$ that is more robust than mesh-based methods and better models sharp and narrow features than volumetric methods.
(3) A method for computing the lifted envelope in $\mathbb{R}^4$ that improves the mesh quality of existing methods for discretizing vector-valued level sets in higher dimensions.

2 Preliminaries

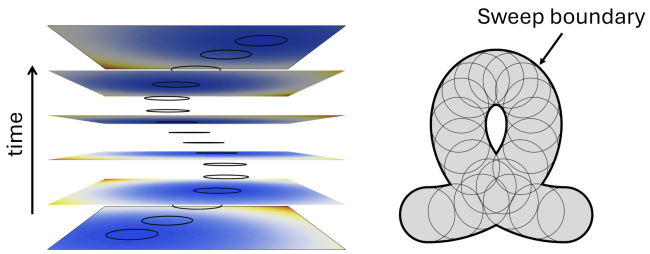
We review the definitions and properties of sweeps that are essential for our paper. More comprehensive and in-depth discussions can be found in classical works such as [Abdel-Malek and Yeh 1997; Blackmore and Leu 1992; Erdim and Ilieş 2007; Martin and Stephenson 1990; Pottmann and Peternell 2000].

2.1 Defining sweeps

We consider a *generalized sweep* (or simply a sweep) in $\mathbb{R}^n$ defined implicitly by a smooth (at least C^2) $(n+1)$-dimensional *sweep function* $f(x,t)$, where x is a point in $\mathbb{R}^n$ and t is a scalar time in the unit range $[0,1]$. The *brush* at time t consists of all points x where $f(x,t) < 0$. The *sweep volume* consists of all points in $\mathbb{R}^n$ that are inside the brush for some $t \in [0,1]$, and the boundary of the sweep volume is known as the *sweep boundary*.

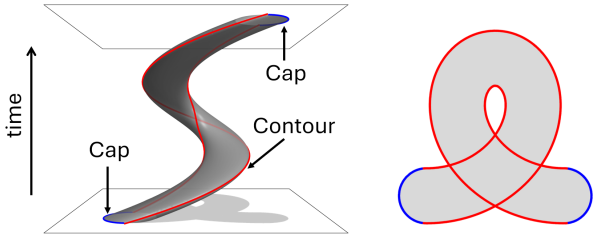
As a special case, consider a "rigid" sweep where the brush shape translates and rotates over time. Let $b(x)$ be the implicit function of the brush at time 0, $T(t)$ and $R(t)$ be the translation vector and rotation matrix as functions of t (that is, a point x at time 0 is moved to location $T(t) + R(t) \cdot x$ at time t). This sweep can be expressed by the following sweep function,

$$f(x,t) = b(R^{-1}(t) \cdot (x - T(t))). \tag{1}$$



(a) Sweep function (b) Sweep volume

Fig. 3. Left: Cross-sections of the 3D sweep function that defines a 2D disk translating over time. Right: The sweep volume (shaded) and boundary (thickened).



(a) Lifted envelope (b) Envelope

Fig. 4. Left: The 3D lifted sweep volume bounded by the sweep set, whose silhouette is the lifted envelope (made up of the contour, bottom and top caps). Right: The lifted envelope projects to the envelope, which contains the sweep boundary (see Figure 3 (b)) as a subset.

Figure 3 illustrates an example of this special case for $n = 2$, where the brush is a circular disk that translates along a curve. Due to the simplicity of this example, we will use it as a running example in our discussions.

In general, the brush may change its shape over time, even altering its topology (e.g., splitting or merging). As a result, our generalized sweep definition is broader than the *sweep with general deformations* [Blackmore et al. 1994], which is limited to brush deformations defined by diffeomorphisms.

2.2 Characterizing sweep boundary

The sweep boundary can be defined implicitly as the zero-level set of the following function,

$$f^*(x) = \min_{t\in[0,1]} f(x,t). \tag{2}$$

This is because x is inside the sweep volume when $f(x,t) < 0$ for some $t \in [0,1]$, which happens if and only if $\min_{t\in[0,1]} f(x,t) < 0$. Note that f^* is continuous but not smooth. In particular, it is not smooth where the minimum is attained by more than one t. The non-smoothness of f^* is manifested on the sweep boundary as sharp features, such as corners for $n = 2$ and crease curves for $n = 3$.

Alternatively, the sweep boundary can be defined as a subset of another structure. Consider the *lifted sweep volume* defined as the set of points $\{x,t\}$ in $\mathbb{R}^{n+1}$ where $f(x,t) < 0$, whose boundary will be called the *sweep set*. Intuitively, the sweep volume in $\mathbb{R}^n$ is the "shadow" of the lifted sweep volume viewed in time, and the sweep boundary in $\mathbb{R}^n$ is the projection of the "silhouette" of the sweep set (see Figure 4 (a)). Points $\{x,t\}$ on this silhouette satisfy $f(x,t) = 0$ (as they are on the sweep set) and additional equality or inequality on the derivative of sweep function in time, $f'(x,t) = \partial f(x,t)/\partial t$, depending on the value of t:

(1) $0 < t < 1$ and $f'(x,t) = 0$.
(2) $t = 0$ and $f'(x,0) > 0$.
(3) $t = 1$ and $f'(x,1) < 0$.

If the sweep is defined by a diffeomorphic map, points satisfying these conditions are known respectively as *grazing*, *ingress*, and *egress* points [Blackmore and Leu 1992]. To be consistent with the nomenclature in this work, we shall call them the *contour*, *bottom cap*, and *top cap*, and their union the *lifted envelope* (Figure 4 (a)).

While every point on the sweep boundary is the projection of some point from the lifted envelope, the reverse is not necessarily true. The projection of the lifted envelope in $\mathbb{R}^n$ is known as the *envelope*, a possibly self-intersecting structure that has the defining property that each point on the envelope is tangent to the brush at some time. The envelope can be similarly defined using Sweep Differential Equations [Blackmore and Leu 1992; Blackmore et al. 1997] or Jacobian rank deficiency [Abdel-Malek et al. 2001; Abdel-Malek and Yeh 1997]. The sweep boundary is the subset of the envelope that is not interior to the sweep volume (see Figure 4 (b)).

2.3 Singularities

Besides self-intersections, the envelope may contain non-smooth features known as *singularities*. These features are projections of points on the lifted envelope with vanishing high-order derivatives. Specifically, an envelope point x is called *singular* if there exists some $t \in (0,1)$ such that $\{x,t\}$ is on the contour part of the lifted envelope (i.e., $f(x,t) = f'(x,t) = 0$) and $f''(x,t) = 0$, where f'' indicates the second order partial derivative in t.

Singularities on the envelope can be further classified by the highest order of vanishing derivatives (if such derivatives exist). A singular point x is called a *cusp* if $f(x,t) = f'(x,t) = f''(x,t) = 0$ but $f'''(x,t) \neq 0$ for some t. Generically, the set of cusp points has dimension $n-2$, such as isolated points in $n = 2$ (Figure 5 (b)) and curves in $n = 3$ (Figure 5 (e)). Note that cusps do not lie on the sweep boundary. This is because the function $f_x(s) = f(x,s)$ has a *stationary point of reflection* at $s = t$. Such points are not local extrema (see Figure 5 (c)), and hence $f(x,s)$ is negative in some neighborhood of $s = t$, implying that x is inside the sweep volume.

We call a singular point x a *pinch* if $f(x,t) = f'(x,t) = f''(x,t) = f'''(x,t) = 0$ but $f''''(x,t) \neq 0$ for some t. The set of pinch points generically has dimension $(n-3)$, such as isolated points in $n = 3$ (Figure 5 (e)). Unlike cusp points, a pinch point could lie on the sweep boundary. This is because $f_x(s) = f(x,s)$ has an *point of undulation* at $s = t$, which is a local extremum (see Figure 5 (f)). We refer readers to classical works such as [Arnold et al. 1986; Whitney 1955] for in-depth discussions on singularities in mappings.

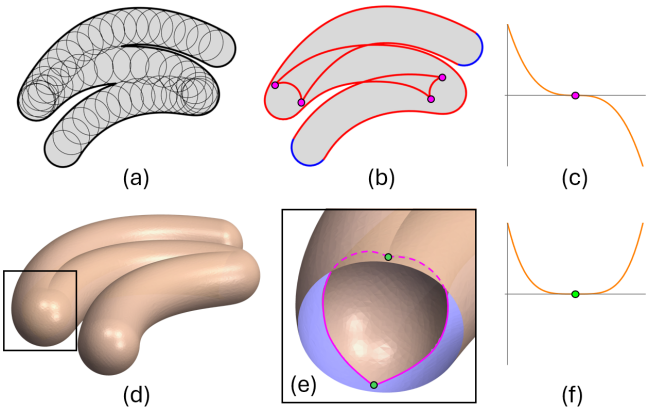


Fig. 5. Top: a 2D sweep (a), its envelope (b) with cusp points (magenta), and a function (c) with a stationary point of inflection (magenta). Bottom: the envelope (d) of the 3D sweep (see Figure 6 (a)), a cut-away view (e) revealing the cusp curves (magenta) and pinch points (green), and a function (f) with a point of undulation (green).

3 Related Works

3.1 Computing sweeps

Sweep volumes have been extensively studied in the past few decades [Abdel-Malek et al. 2006]. We briefly review representative computational methods with an eye towards their robustness, generality, and ability to approximate geometric and topological features.

3.1.1 Volumetric methods. These methods discretize the sweep boundary on a volumetric grid to separate grid points that are inside and outside the sweep volume. To determine whether a grid point x is inside, many methods follow the implicit definition of the sweep boundary (Equation 2) and compute the signed value $f^*(x)$ either by discrete sampling (known as *stamping*) [Himmelstein et al. 2009; Schroeder et al. 1994; Sourin and Pasko 1995; von Dziegielewski et al. 2010] or root-finding [Sellán et al. 2021; Sourin and Pasko 1995]. While stamping introduces either aliasing artifacts (at low grid resolutions) or large computational cost (at high resolutions), root-finding produces smoother surfaces and can be made efficient using tracing techniques such as space-time continuation [Sellán et al. 2021]. Alternatively, a discrete inside/outside label at x can be obtained by either intersecting the reverse sweep trajectory at x with the brush at time 0 [Erdim and Ilieş 2008, 2010; Ilies and Shapiro 1999; Rossignac et al. 2007] or propagating the labels from the outside towards an initial surface (as part of a mesh-based method) [Kim et al. 2003; Peternell et al. 2005; Von Dziegielewski et al. 2013; Zhang et al. 2009]. However, the former is limited to sweeps with reversible trajectories, and the latter cannot recover interior voids.

A common drawback of volumetric methods is their limited ability to model fine geometric features, such as sharp creases and closed-by surface parts, due to a finite grid resolution. They tend to create rounded creases and artifacts near narrow spaces, as shown in Figures 2 (c) and 6 (c). While using finer grids can improve the sharpness and reduce artifacts, these issues never truly go away, and this comes at the cost of significantly increased compute time and output size, as shown in Figures 2 (d) and 6 (d). While feature-sensitive iso-surfacing methods can be used, such as Dual Contouring [Ju et al. 2002], they require additional information (e.g., normals) that can be difficult to obtain precisely for sweep boundaries.

3.1.2 Mesh-based methods. These methods work by creating a set of candidate surfaces, which contain the sweep boundary, then trimming away the unwanted parts. We will separately discuss methods that address each task.

A natural choice of the candidate surface is the *envelope*, as reviewed in Section 2.2. Since the envelope is described by the solution of non-linear equations [Martin and Stephenson 1990], exact computation is only possible for simple brush shapes and sweep motions [Adsul et al. 2014, 2015; Kieffer and Litvin 1991; Machchhar et al. 2017; Rabl et al. 2008; Wang and Wang 1986]. Even approximating the envelope is challenging, due to its self-intersecting structure and singularities (see Section 2.3). Many methods construct the envelope from points on the moving brush that are tangent to the envelope, known as *characteristic curves*. These methods either sample the curves as scattered points and then perform point-based surface reconstruction [Blackmore and Leu 1992; Peternell et al. 2005], or directly connect curves from adjacent time points into triangle strips [Madrigal and Joy 1999; Yang and Abdel-Malek 2005] or skinned parametric surfaces [Rossignac et al. 2007; Weinert et al. 2004; Xu et al. 2007]. However, the connecting approach fails on complex sweeps where the characteristic curves change significantly in shape and topology over time.

Another type of candidate surface is the union of sweeps of the boundary elements (e.g., edges and triangles) of the brush, which is also known to contain the sweep boundary [Weld and Leu 1990]. Current methods for computing such candidate surfaces, however, are limited to specific types of sweeps, such as a triangle mesh undergoing translation and rotation [Abrams and Allen 1995; Campen and Kobbelt 2010; Kim et al. 2003; Von Dziegielewski et al. 2013; Zhang et al. 2009] or a trivariate B-spline solid undergoing a parametric transformation [Conkey and Joy 2000].

To trim away parts of the candidate surface that lie inside the sweep volume, many methods employ front-propagation over the arrangement of the surface [Abrams and Allen 1995; Campen and Kobbelt 2010] or a volumetric grid [Kim et al. 2003; Peternell et al. 2005; Von Dziegielewski et al. 2013; Zhang et al. 2009], starting from the outside of the sweep boundary. However, such propagation cannot reach interior voids, which are typically missing in the output. Alternatively, point-wise classification (e.g., based on intersecting reverse trajectories with the brush) can be applied to prune points or patches of the candidate surface [Blackmore and Leu 1992; Blackmore et al. 1994, 1997, 1999; Weld and Leu 1990; Yang and Abdel-Malek 2005]. However, naive pruning may result in gaps and self-intersections on the output surface.

In summary, it remains challenging for mesh-based methods to produce a water-tight, intersection-free sweep boundary with interior voids for general brush shapes and motions.

3.1.3 Data driven methods. These methods learn to predict aspects of a sweep, such as the volume measure [Chiang et al. 2020], or a geometric representation of the sweep volume of a rigid sweep [Marschner et al. 2023] or between a pair of robot configurations [Baxter et al. 2020]. However, their success is limited to the range of

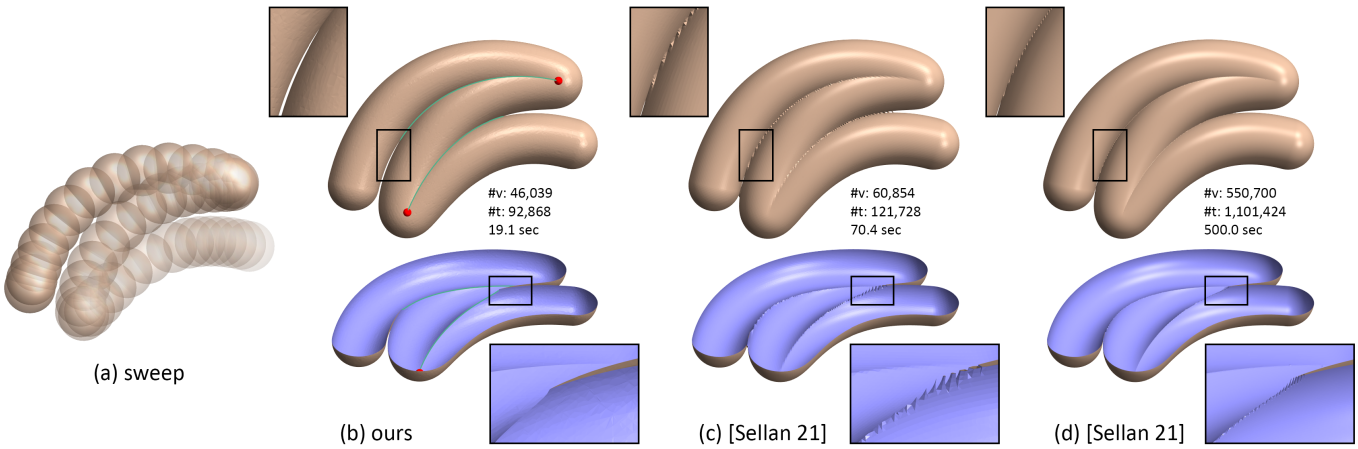


Fig. 6. Comparing sweep boundaries computed by our method (b) and by [Sellán et al. 2021] at grid cell size 0.003 (c) and 0.001 (d) for a translating sphere (a). Boxes highlight narrow regions on the boundaries, where our method produces fewer artifacts and sharper creases.

sweeps (e.g., robot type or brush shape) in the training set, making them difficult to generalize to arbitrary brushes and motions.

3.2 Level set intersection in higher dimensions

The core of our method computes the intersection of the level sets of two functions in 4D. While discretizing the level set of a scalar function in 2 and 3 dimensions has been extensively studied [De Araújo et al. 2015], much less work exists in higher dimensions or for multiple functions. Current methods for computing the level set of a scalar function in $\mathbb{R}^n$ extend standard techniques in 3D, such as Marching Cubes and Marching Tetrahedra, to work on a decomposition of $\mathbb{R}^n$ into hypercubes [Bhaniramka et al. 2004; Castelo et al. 2022] or simplices [Schwaha and Heinzl 2010]. These methods produce a $(n-1)$-manifold represented as a polyhedral or simplicial complex.

The intersection of the level sets of k functions is generically a $(n-k)$-manifold. One approach to compute it is to decompose $\mathbb{R}^n$ into simplices and take the union of the intersection of k hyperplanes within each simplex [Allgower and Schmidt 1985; Boissonnat et al. 2023; Reia et al. 2025]. While efficient, this method can only achieve linear accuracy within each simplex. Alternatively, one may iteratively apply the method for computing the simplicial level set of a scalar function over a simplicial domain [Min 2003; Weigle and Banks 1996]. Starting from the level set of the first function, this approach iteratively computes the intersection of the first i functions, for $i = 2, \ldots, k$, by sampling the i-th function and computing its simplicial level set on the intersection of the previous $(i-1)$ functions. The iterative approach achieves higher accuracy as it evaluates functions at additional samples created by repeated intersections. However, this comes at the cost of a larger number of simplices and lower simplex quality with increasing k. Our method takes the iterative approach but uses a different method to compute the level set of the first function in $\mathbb{R}^4$, leveraging its unique properties to lower the number of triangles on the final 2-manifold while improving their quality.

Efficient deployment of grid-based discretization requires a grid that adapts its resolution to the geometry and topology of the level set. While adaptive grid generation has been well-studied for discretizing level sets in low dimensions (2 and 3), relatively few methods are designed for discretizing the intersections of level sets. Most of these methods are either specialized to low-degree polynomials [Dupont et al. 2007; Mourrain et al. 2005; Schömer and Wolpert 2006] or require interval analysis [Allgower and Georg 1989; Boissonnat and Wintraecken 2022; Diatta et al. 2019], making them unsuited for general functions (e.g., not in analytical form). The recent method of [Ju et al. 2024], based on iterative refinement of a simplicial grid, is the only method we know that can be applied to general functions in any dimension. We build on it to design an adaptive grid generation scheme that is tailored to our computation of the intersection of two 4D level sets.

4 Theoretical foundation

Before diving into our method over the next few sections, we first discuss the theoretical insights that guide our method. Like many mesh-based methods, we obtain the sweep boundary by computing and trimming the envelope. To make the process general and robust, we make two observations of an envelope in $\mathbb{R}^n$:

(1) The lifted envelope, whose projection is the envelope, lies at the intersection of the zero-level sets of two implicit functions in $\mathbb{R}^{n+1}$ (Section 4.1).
(2) The winding number of a properly oriented envelope completely determines the subset of the envelope that forms the sweep boundary (Section 4.2).

The first observation allows the lifted envelope (and in turn, the envelope) to be robustly discretized by iso-surfacing. The second observation enables simple and robust trimming of the envelope by leveraging existing tools for computing mesh arrangement and winding numbers. Both observations apply to a generalized sweep in any dimension, which we shall detail next.

Our discussion assumes that the sweep function $f : \mathbb{R}^{n+1} \to \mathbb{R}$ is *generic* (i.e., in general position). Specifically, recall that s a regular

value of a differentiable scalar function h if the s-level set of h avoids the critical points of h. We ask that 0 is a regular value of (i) f, (ii) the time derivative function f', (iii) f' restricted to the zero-level set of f, and (iv) each of (i,ii,iii) restricted to the hyperplanes at $t = 0$ and $t = 1$. While conditions (i,ii) imply that the zero-level sets of f and f' are both n-manifolds, (iii) ensures that their intersection (i.e., the contour) is an $(n-1)$-manifold, and (iv) further implies that the union of the contour and caps (i.e., the lifted envelope) is an $(n-1)$-manifold. Furthermore, we assume that the lifted envelope has a generic projection (i.e., the envelope), whose self-intersections and singularities (see Section 2.3) have dimension $n-2$.

4.1 Implicit representation of lifted envelopes

As reviewed in Section 2, the lifted envelope in $\mathbb{R}^{n+1}$ consists of three parts, the contour, the bottom cap and the top cap. Each part is the intersection of the sweep set (i.e., the zero-level set of the sweep function $f(x,t)$) and some geometry implicitly defined by the time t and the time derivative function $f'(x,t)$. Specifically,

- The contour lies on the zero-level set of $f'(x,t)$ in the range $0 < t < 1$, which we call the *contour set*.
- The bottom cap lies on the zero-superlevel set of $f'(x,t)$ at $t = 0$, which we call the *bottom cap set*.
- The top cap lies on the zero-sublevel set of $f'(x,t)$ at $t = 1$, which we call the *top cap set*.

Since f is C^2-continuous, and by our genericity assumptions, all three sets are C^1-continuous n-manifolds with boundaries. In addition, they share common boundaries defined by $f'(x,t) = 0$ and $t = 0$ or $t = 1$, which are $(n-1)$-manifolds. Therefore, the union of the contour set and the two cap sets is a closed, piecewise smooth n-manifold, which we call the *silhouette set* (see Figure 7 (a)). Note that the silhouette (resp. contour, bottom cap, or top cap) set contains the lifted envelope (resp. contour, bottom cap or top cap) of any sweep function $h(x,t) = f(x,t) + c$ for $c \in \mathbb{R}$.

We can replace the piecewise definition of the silhouette set above by the zero-level set of a *single* function, which will be more convenient for analysis and computation. To do so, we define a *silhouette function* g that extends f' beyond the time range of $[0,1]$,

$$g(x,t) = \begin{cases} f'(x,t), & 0 \le t \le 1 \\ 1, & t > 1 \\ -1, & t < 0 \end{cases} \tag{3}$$

Since g is not continuous at $t = 0$ and $t = 1$, we define its zero-level set as the boundary of (the closure of) its zero-superlevel set. It is easy to verify that the zero-level set of g coincides with the silhouette set defined above. This definition also shows that the silhouette set is orientable, as it partitions $\mathbb{R}^{n+1}$ into two regions with different signs of g.

To summarize, the lifted envelope is the intersection of two implicit surfaces in $\mathbb{R}^{n+1}$, the zero-level set of the sweep function f (a.k.a. the sweep set) and the zero-level set of the silhouette function g (a.k.a. the silhouette set). This is illustrated in Figure 7 (b).

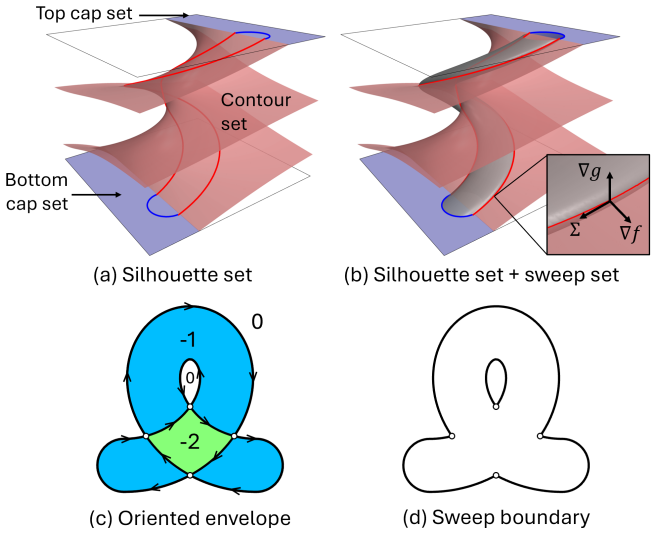


Fig. 7. (a) The silhouette set of a 2D sweep (Figure 3) consisting of the contour set (red), the bottom and top cap sets (blue). (b) The lifted envelope as the intersection of the silhouette set and sweep set (gray). The insert shows an outward tangent basis associated with a point on the lifted envelope. (c) The outward-oriented envelope and the winding numbers in each connected region. (d) The sweep boundary as the subset of the envelope bounding regions with 0 winding.

4.2 Sweep boundary from winding numbers

We next introduce a combinatorial characterization of the sweep boundary using the winding numbers of the envelope. The characterization depends on a carefully chosen orientation of the envelope, which we discuss first.

4.2.1 Orienting the envelope. Consider a point p on the smooth portion of the $(n-1)$-dimensional lifted envelope in $\mathbb{R}^{n+1}$ (i.e., p is not on the boundaries shared by the caps and the contour). An orientation of the lifted envelope at p can be expressed by a basis of $n-1$ tangent vectors, which we denote by Σ. Since the lifted envelope lies simultaneously on the level sets of the sweep function f and silhouette function g, Σ is orthogonal to both of their gradients, denoted by ∇f and ∇g (if p is on the cap, where g is not continuous, we direct ∇g to point in the positive time direction). Our genericity assumptions ensure that ∇f and ∇g are linearly independent, regardless of whether p is on the contour or a cap, which implies that $\{\Sigma, \nabla f, \nabla g\}$ forms a basis of $\mathbb{R}^{n+1}$. We say that the tangent basis Σ is *outward* if $\{\Sigma, \nabla f, \nabla g\}$ forms a positively oriented basis; that is,

$$\det(\{\Sigma, \nabla f, \nabla g\}) > 0.$$

As a special case, consider $n = 2$. The lifted envelope is a curve in 3D, and a tangent basis Σ consists of a single tangent vector along the curve. The tangent vector is outward if it forms a right-handed basis with ∇f and ∇g, as shown in the insert of Figure 7 (b). In higher dimensions, the outward rule can be intuitively stated as follows. Consider the n-dimensional hyperplane Γ in $\mathbb{R}^{n+1}$ that is tangent to the level set of g (i.e., the silhouette set) at point p and whose normal points towards the side where g is positive. An outward

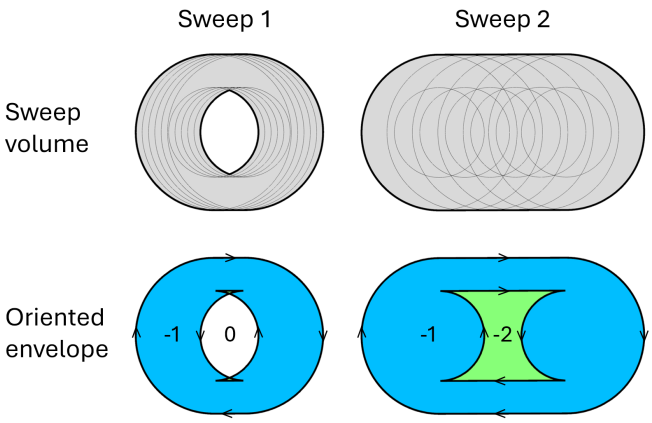


Fig. 8. Sweeping an annulus along a shorter (left) and longer (right) path, showing the sweep volumes (top) and the outward-oriented envelopes (bottom) with winding numbers.

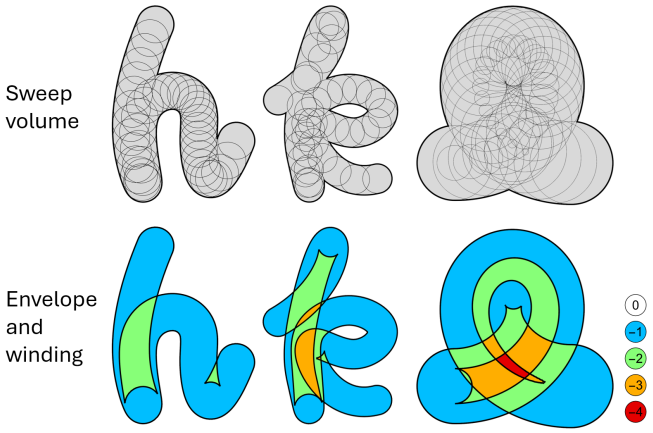


Fig. 9. 2D sweeps with complex trajectories (letters "h" and "k") or topological changes to the brush (right, from an annulus to a disk).

basis Σ defines an $(n-1)$-dimensional hyperplane in Γ, tangent to the level set of f (i.e., the sweep set), whose normal points towards the side where f is positive. This rule will be used to guide our discrete algorithm (see Section 6).

We can similarly define an outward orientation of the envelope by projecting the outward tangent bases from the lifted envelope. We call an envelope point *regular* if it is away from the singularities and self-intersections. Our genericity assumptions ensure that most points of the $(n-1)$-dimensional envelope are regular, since the singularities and self-intersections only have $n-2$ dimensions. For each regular envelope point x, there exists a unique time, t_x, such that $\{x, t_x\}$ is on the lifted envelope. Let Σ be a tangent basis at $\{x, t_x\}$, and $\overline{\Sigma}$ its projection into $\mathbb{R}^n$ by removing the $(n+1)$th coordinate in each basis vector. Then $\overline{\Sigma}$ is a tangent basis of the envelope at x. We say that $\overline{\Sigma}$ is *outward* if Σ is outward, and the envelope is *outward-oriented* if each point is associated with an outward tangent basis. See Figure 7 (c) for an example of an outward-oriented envelope.

4.2.2 Defining the sweep boundary. With the orientation of the envelope defined, we can now introduce our characterization of the subset of the envelope that is the sweep boundary. Recall that the *winding number* of a closed, oriented curve C in the plane around a point x is the total number of times C travels counterclockwise around x. The winding number can be generalized to higher dimensions based on the topological degree of a map [Outerelo et al. 2009]. Specifically, given a closed and oriented $(n-1)$-manifold C in $\mathbb{R}^n$, the winding number of C around a point x is the degree of the continuous map from C_x, the radial projection of C onto the unit n-sphere centered at x, to the unit sphere itself. The winding number is always an integer and is constant for all points in a connected component of space away from C. Our key observation is that,

Theorem 4.1. *The sweep boundary is the subset of the outward-oriented envelope that bounds the space with winding number 0.*

The proof is given in Appendix A. The key idea is to relate the winding number to the number of disjoint time intervals during which a point is inside the brush. We call such intervals *layers*, as in layers of brush strokes. We show that the number of layers, like the winding number, is constant in each connected space away from the envelope, changes by ±1 as a point moves across the envelope, and the sign is determined by the direction of movement relative to the outward orientation of the envelope. The winding numbers and the resulting sweep boundary are illustrated on our running example in Figure 7 (c,d).

The outward orientation of the envelope is essential for our winding-number-based characterization of the sweep boundary. In the two sweeps shown in Figure 8, the envelope contains an outer curve and an inner curve in both cases, but only the sweep on the left has a void. In both examples, the zero-winding cells under the outward orientation correctly identify the sweep boundary. We demonstrate our characterization on a few more examples in Figure 9 where the brush either travels along a complex trajectory or changes topology during sweeping. Such sweeps are challenging for existing methods that classify envelope points by computing reverse trajectories and intersecting them with the brush [Erdim and Ilieş 2008, 2010; Ilies and Shapiro 1999; Rossignac et al. 2007].

5 Method overview

The observations in the previous section suggest a two-stage method for computing a discrete sweep boundary in $\mathbb{R}^n$, combining elements from both volumetric and mesh-based methods:

- Stage 1 (*volumetric*): Compute the outward-oriented envelope by computing the lifted envelope in $\mathbb{R}^{n+1}$ as the intersection of two level sets.
- Stage 2 (*mesh-based*): Compute the arrangement of the projected envelope in $\mathbb{R}^n$ and output the subset bounding cells with zero winding.

For the case of $n = 3$, the second stage can be easily implemented by leveraging existing methods for computing mesh arrangement and winding numbers, such as [Zhou et al. 2016], with simple post-processing. While these methods are numerically robust, they may produce numerous spurious arrangement cells around the envelope

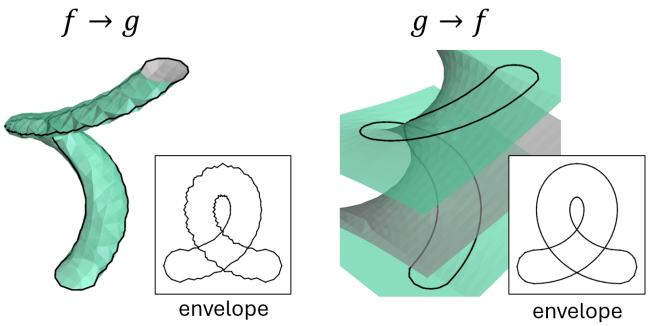


Fig. 10. Computing the 2D envelopes by first extracting the iso-surface (green) of f (right) or g (left) on the same tetrahedral grid and then computing the iso-curve of the other function on the iso-surface.

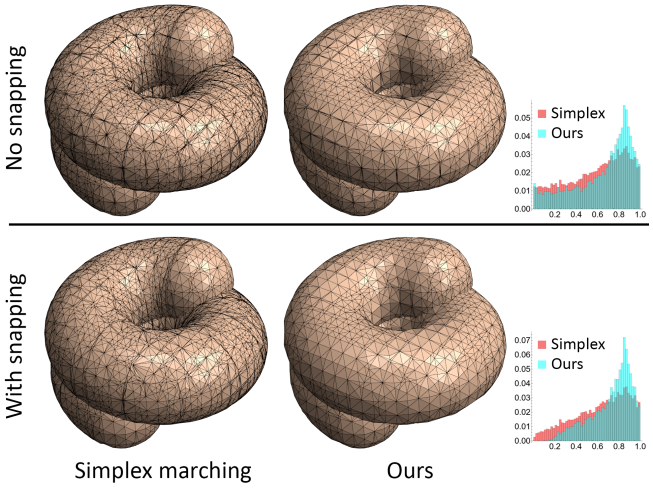


Fig. 11. The triangulated envelopes computed by iterative application of simplex-marching (left) and our method (middle), without (top) and with (bottom) grid snapping, on the same uniform simplicial grid in 4D. Histograms show triangle quality as the normalized area-length ratio.

singularities, where the intersecting triangles are nearly tangent. We prune cells whose volume is smaller than a threshold. We also extract the crease curves, where multiple arrangement patches meet, and crease points at the ends or junctions of crease curves. They are shown as green curves and red points in all our examples.

Our main algorithmic contribution lies in the first stage. We will first introduce a grid-based algorithm for discretizing the intersection of two level sets in 4D, tailored to the lifted envelope (Section 6). We will follow up with a method to generate a grid with adaptive resolution, in both space and time, through iterative refinement (Section 7). Our method works for any "black-box" sweep function $f(x,t)$ (satisfying the genericity assumptions in Section 4) that provides the value and gradient at a query location x and time t.

6 Computing lifted envelopes

We describe an algorithm for computing a lifted envelope in $\mathbb{R}^4$ equipped with an outward orientation. Recall that the lifted envelope is the intersection of the sweep set, which is the zero-level set of the sweep function f, and the silhouette set, which is the zero-level set of the silhouette function g (see Figure 7 (b)).

6.1 Motivation

A common approach for discretizing an intersection of two level sets is to first compute the level set of one function, which produces a codimension-1 manifold $\mathcal{M}$, and then sample the other function *on* $\mathcal{M}$ and compute the level set, which produces a codimension-2 manifold [Min 2003; Weigle and Banks 1996]. Our algorithm follows this two-step approach but differs from existing methods in two ways, as explained below.

6.1.1 Order of functions. While existing methods make an arbitrary choice of which function to use in each step, the order of functions ($f \to g$ versus $g \to f$) matters for our problem. As discussed in Section 2.2, the lifted envelope lies at the "silhouette" of the sweep set as viewed in the time direction (see Figure 4 (a)). However, discrete surfaces often give rise to jagged silhouettes [Liu et al. 2023]. This is demonstrated on our 2D running example in Figure 10 (left), where we follow $f \to g$ and compute $\mathcal{M}$ as the sweep set, the iso-surface of f (using Marching Tetrahedron on a uniform tetrahedral grid). Observe that its silhouette, or the iso-curve of g on $\mathcal{M}$, is far from being smooth, and neither is its projection, the envelope.

To extract smoother envelopes, we observe that points on the lifted envelope tend to *avoid the silhouette of the silhouette set*. Such a point $\{x,t\}$ must satisfy $f''(x,t) = 0$, implying that x is a singularity of the envelope. By our genericity assumptions, envelope singularities have a lower dimensionality ($n-2$) than the envelope ($n-1$). Furthermore, as discussed in Section 2.3, the only type of singularities that may lie on the sweep boundary (i.e., pinch points) is even more rare, generically two dimensions lower than the envelope. Guided by the observation, we choose to first compute $\mathcal{M}$ as the silhouette set and then the level set of f restricted to $\mathcal{M}$. This leads to smoother discretization of the lifted envelope, particularly the part that projects to the sweep boundary, as demonstrated in Figure 10 (right).

6.1.2 Mesh element quality. To discretize a zero-level set in $\mathbb{R}^4$, existing methods typically decompose the space into 4D simplices and extract the discrete iso-(hyper)surface within each simplex. This results in a collection of 3D simplices (i.e., tetrahedra), which are then used to compute the triangulated iso-surface of another function (e.g., by Marching Tetrahedra). However, 4D iso-surfacing using simplices may create tetrahedra with poor shape quality, such as those with short edges and small dihedral angles, which in turn leads to low-quality triangles after 3D iso-surfacing. We demonstrate this issue on a rigid sweep example (a sphere moving along a spiral) in Figure 11 (top-left) using a uniform 4-simplex grid. Note that the mesh produced by 4D and 3D iso-surfacing in this way contains numerous near-degenerate triangles (see also the triangle quality histogram on the far right). Such triangles may lead to numerical issues in downstream tasks, including computing the arrangement.

One way to improve the quality of iso-surface triangles extracted on a 3D grid is by "snapping" the grid vertices to the iso-surface.

While this approach can eliminate near-degenerate triangles extracted on a well-shaped tetrahedral grid [Labelle and Shewchuk 2007], it is less effective if the grid tetrahedra have poor shapes themselves, like those produced by iso-surfacing over 4D simplices. As shown in Figure 11 (bottom-left), while snapping reduces the number of low-quality triangles, many degenerate triangles remain.

We will describe a new 4D iso-(hyper)surfacing algorithm that produces mostly well-shaped tetrahedra. Our algorithm leads to better-shaped triangles after 3D iso-surfacing and grid snapping, as shown in Figure 11 (middle).

6.2 Overview

Instead of simplices, we decompose $\mathbb{R}^4$ into 4D *columns*, each extruding in the time direction a tetrahedron in a given 3D tetrahedral grid. The choice is motivated by our observation made above that the lifted envelope tends to avoid where the silhouette set "folds" in time (i.e., its own silhouette). As a result, the part of the silhouette set containing the lifted envelope is more likely to intersect "transversely" with a column, meaning that the intersection is a collection of 3D tetrahedral cross-sections of the column. As the 3D projection of such a cross-section is the tetrahedron from which the column is extruded, and assuming the given tetrahedral grid is well-shaped, we can obtain mostly well-shaped tetrahedra by using the columns as the units for 4D iso-surfacing.

Specifically, our algorithm takes in a "3.5D" representation of a 4D grid, which consists of a 3D tetrahedral grid and a sequence of *time stamps* at each grid vertex (grid generation will be discussed in the next section). We proceed in two steps to compute the lifted envelope:

(1) *Marching columns* (Section 6.3): For each tetrahedron and the time stamps at its vertices, compute the discrete zero-level set of the silhouette function g within the column as a set of polyhedra (mostly tetrahedra).
(2) *Marching polyhedra* (Section 6.4): For each polyhedron generated in the previous step, compute the discrete zero-level set of the sweep function f as triangles.

To ensure the outward orientation of the lifted envelope, and following the rule stated in Section 4.2, we orient each polyhedron (resp. triangle) towards the part of the column (resp. polyhedron) where g (resp. f) is *positive*.

We illustrate our algorithm in 2D on our running example; see Figure 12. The input is a uniform triangular grid with uniform time samples at each grid point. Each column takes the shape of a prism that extrudes a triangle in time. Column-marching produces oriented polygons in each column that collectively form a consistently oriented polygonal surface in 3D, approximating the silhouette set (see (a)). Note that the majority of polygons are triangles (colored green/gray on the front/back sides) whose projections in 2D are grid triangles, with only a few multi-sided polygons (colored red on both sides) where the silhouette set "folds" and hence intersects the columns non-transversely. Polyhedron-marching (or rather, polygon-marching in 2D) on this polygonal surface produces a closed polyline that approximates the lifted envelope (see (b)). Note that the lifted envelope largely avoids the multi-sided polygons.

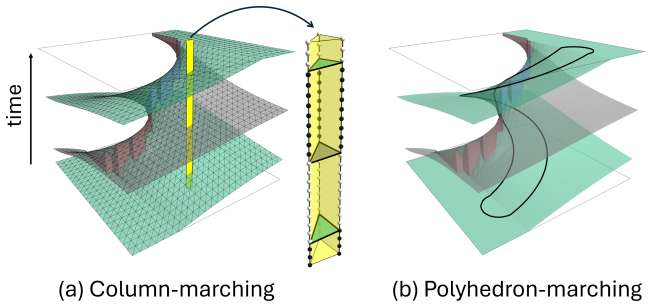


Fig. 12. Computing the lifted envelope of a 2D sweep. (a): The silhouette set (a surface in 3D), consisting of triangles (green/gray on the front/back) and multi-sided polygons (red/blue), computed by marching 3D columns extruded from 2D triangles on a grid. One column (yellow) is highlighted, showing the time stamps (dark/light balls for negative/positive g) and the extracted silhouette polygons. (b): The lifted envelope (black curve) computed by marching the silhouette set polygons.

6.3 Marching columns

To extract the polyhedral iso-surface of g in the 4D column of a tetrahedron s, we visit elements of s at increasing dimensionality and produce polyhedral elements at the corresponding dimensionality. To avoid ambiguity, we shall call elements of s *t-vertices*, *t-edges*, and *t-faces*, and polyhedral elements *p-vertices*, *p-edges*, and *p-faces*. We shall refer to the extrusion of a t-vertex, t-edge, and t-face in time as a *timeline*, *time-quad*, and *time-prism*. We assume that s is positively oriented; that is, each t-face is oriented towards the remaining t-vertex. We proceed in the following steps.

(1) For each t-vertex x, create p-vertices at the intersections of the silhouette set with the timeline of x. This is done by evaluating $f'(x,t)$ at the time stamps of x and computing the *zero-crossing times* between successive time stamps with opposite signs by linear interpolation. Recall that g extends $f'(x,t)$ by -1 for all $t < 0$ and 1 for all $t > 0$ (Equation 3). Accordingly, we add a zero-crossing time at 0 if $f'(x,0) > 0$ and at 1 if $f'(x,1) < 0$ (see Figure 13 (a)). As such, there are always an *odd* number of zero-crossings on the timeline. We create one p-vertex $\{x,t\}$ for each zero-crossing time t. Intuitively, these p-vertices divide the timeline of x into time intervals with alternating signs of g.
(2) For each directed t-edge $\{x,y\}$, connect the p-vertices on the timelines of x and y into directed p-edges. Since each timeline has an odd number of p-vertices, their total number is even. We sort the p-vertices by their time values and connect every consecutive pair into a p-edge. Intuitively, these p-edges partition the time-quad of $\{x,y\}$ into regions, such that each region connects overlapping time intervals on the two timelines with the same sign of g (see Figure 13 (b)). We say a region is positive (resp. negative) if it connects positive (resp. negative) time intervals. We direct each p-edge so that the region on its left is positive when the time-quad is drawn on the left of the directed edge $\{x,y\}$.
(3) For each oriented t-face $\{x,y,z\}$, form directed cycles from the directed p-edges on the time-quads of t-edges $\{x,y\}$,

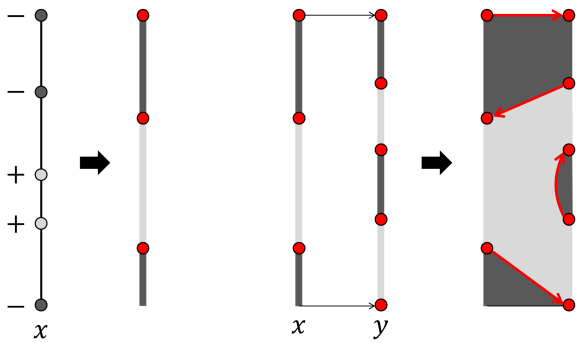


Fig. 13. Processing a t-vertex (a) and a t-edge (b) in column-marching. Light (resp. dark) dots, segments, and polygons are positive (resp. negative) time samples, timeline intervals, and time-quad regions. Red dots and arrows are p-vertices and directed p-edges.

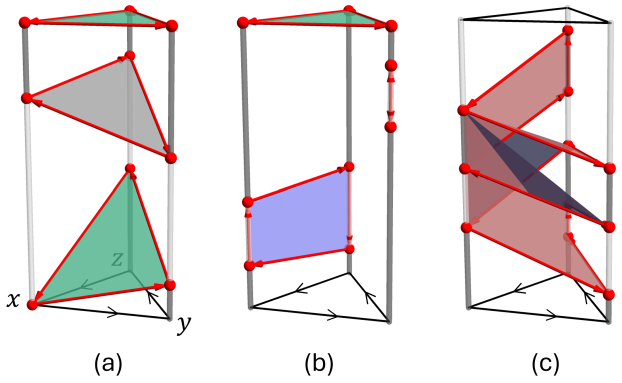


Fig. 14. Examples of processing a t-triangle in column-marching. Red dots and arrows are p-vertices and p-edges obtained by vertex and edge processing (see Figure 13). A p-face may be triangular (green/gray on the front/back) or multi-sided (red/blue on the front/back). The cycle made up of two identical p-edges with opposite directions in (b) does not yield a p-face.

$\{y, z\}$, and $\{z, x\}$. Each cycle with three or more p-edges becomes a p-face (see Figure 14). Non-triangular p-faces arise when the silhouette set intersects the column non-transversely (e.g., Figure 14 (b,c)). The directions of the p-edges ensure that the p-faces are oriented towards the space in the time-prism of $\{x, y, z\}$ where g is positive.

(4) Identify connected components of p-faces in the time-prisms of all t-faces of s. Each connected component containing three or more p-faces becomes a polyhedron.

The column-marching procedure guarantees that the output polyhedral mesh is topologically manifold at the polyhedral faces (i.e., p-faces), in the sense that each p-face a is incident to exactly two polyhedra unless a is at the domain boundary. These two polyhedra correspond to the two tetrahedra in the input grid sharing a t-face b whose time-prism contains a, unless b is on the boundary of the tetrahedral grid. While a polyhedron may be geometrically self-intersecting (e.g., the spiral-like p-face in Figure 14 (c)), this poses no problem to our method as all self-intersections on the envelope will be eventually resolved by computing the arrangement.

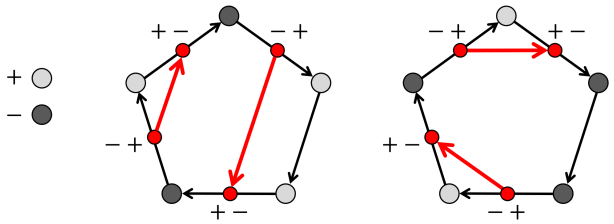


Fig. 15. Connecting zero-crossings (red dots) on directed edges (black arrows) of a polyhedral face into directed segments (red arrows) during polyhedral marching, showing two opposite sign configurations at the polyhedral vertices (light or dark gray dots). Each directed segment starts at a −+ zero-crossing and ends at a +− zero-crossing.

6.4 Marching polyhedra

To extract the triangulated iso-surface of f inside a polyhedron p, we follow the standard tracing-based approach while taking care to orient the resulting triangles towards the space in the polyhedron where f is positive.

First, we compute a zero-crossing of f on each edge of p (i.e., a p-edge). To improve surface smoothness, we approximate f along a p-edge by cubic Hermite interpolation of the values and gradients of f at the p-vertices, and then locate its root using Halley's method. Next, zero-crossings on each oriented p-face are connected into direct segments. Specifically, given a directed p-edge $\{a, b\}$ with a zero-crossing, we label the zero-crossing as −+ (resp. +−) if $f(a) <$ 0 (resp. $>$ 0). We trace the ordered p-edges bounding the p-face and form a segment from each −+ zero-crossing to the next +− zero-crossing (see two examples in Figure 15). Then, the directed segments on all p-faces of p are connected into directed cycles, which are triangulated if the cycle has more than three segments. Finally, the triangles are projected to 3D by dropping the 4th coordinate from each zero-crossing.

The polyhedra generated by column-marching, particularly those around the lifted envelope, are mostly tetrahedra identical to those in the input grid. Polyhedral marching in these tetrahedra reduces to the standard Marching Tetrahedra. We use grid-snapping [Labelle and Shewchuk 2007] (with snapping threshold 0.1 of the edge length) to remove near-degenerate triangles and improve overall triangle quality, as shown in Figure 11 (middle).

7 Adaptive grid generation

The surfacing algorithm described in the previous section relies on an input grid that samples both space and time. Due to the high dimensionality, uniform sampling could easily lead to an excessive grid size that is expensive to store or process. To make our method practically useful, it is important to be able to adapt the sampling density and only use more samples where we need them; that is, around the lifted envelope in $\mathbb{R}^4$, and particularly its subset that projects to the sweep boundary in $\mathbb{R}^3$. In addition, the grid should

contain well-shaped tetrahedra around the sweep boundary, so that surfacing produces well-shaped triangles.

We describe an adaptive grid generation algorithm that meets these demands. Our algorithm heavily borrows from the work by [Ju et al. 2024], which employs iterative refinement to create an adaptive, simplicial grid with well-shaped simplices for the purpose of discretizing the intersection of multiple implicit surfaces. We tailor their algorithm to work with our 3.5D grid representation and the two-step surfacing algorithm.

7.1 Iterative refinement

The grid is generated in a coarse-to-fine manner. Starting from a coarse, initial grid that consists of a uniform tetrahedral mesh and uniform time stamps at each vertex, we refine the grid both spatially and temporally. We adopt the longest-edge bisection approach for spatial refinement [Plaza and Rivara 2003; Rivara 1991]. Specifically, given a tetrahedron whose longest edge is $\{x, y\}$, we insert a new vertex z at the midpoint of $\{x, y\}$, which splits every tetrahedron incident to the edge into two smaller tetrahedra. The time stamps at z are defined as the union of those on x and y. Similarly, for temporal refinement, we split a time interval between two time stamps at a vertex by inserting a new time stamp at the middle of the interval.

Spatial and temporal refinements are invoked iteratively. We prioritize temporal refinement, whenever possible, because the time samples take less storage and their addition does not affect the tetrahedral mesh structure. Entities that need to be refined are kept in two queues, a *spatial queue* of tetrahedra and a *temporal queue* of time intervals. To yield good tetrahedral quality, we follow [Ju et al. 2024] and sort all tetrahedra in the spatial queue by descending lengths of their longest edges. Similarly, intervals in the temporal queue are sorted by descending lengths.

At each iteration, the algorithm pops an interval from the temporal queue, if it is not empty, or a tetrahedron from the spatial queue otherwise, and performs a temporal or spatial refinement. Tetrahedra affected by the refinement, such as the newly created tetrahedra in spatial refinement and those incident to the vertex where a time interval is refined, are then checked by a set of *refinement criteria*, to be explained below. The criteria decide if a tetrahedron, or a time interval at one of its vertices, should be refined, and the refinable entity is added to the corresponding queue. The algorithm finishes when both queues are empty or some termination condition is met. In our implementation, we stop processing the temporary (resp. spatial) queue if the longest interval (resp. edge length) in the queue is smaller than a threshold ϵ_{time} (resp. ϵ_{space}).

7.2 Refinement criteria

Given a tetrahedron s, we need to decide if s or the time intervals at its vertices need to be refined. As our goal is to prioritize samples around the part of the lifted envelope that projects to the sweep boundary, our criteria consist of the following ordered list of checks and actions:

(1) Is s inside the sweep volume? If so, no refinement is needed. Otherwise,
(2) Does the 4D column of s intersect the lifted envelope? If not, no refinement is needed. Otherwise,

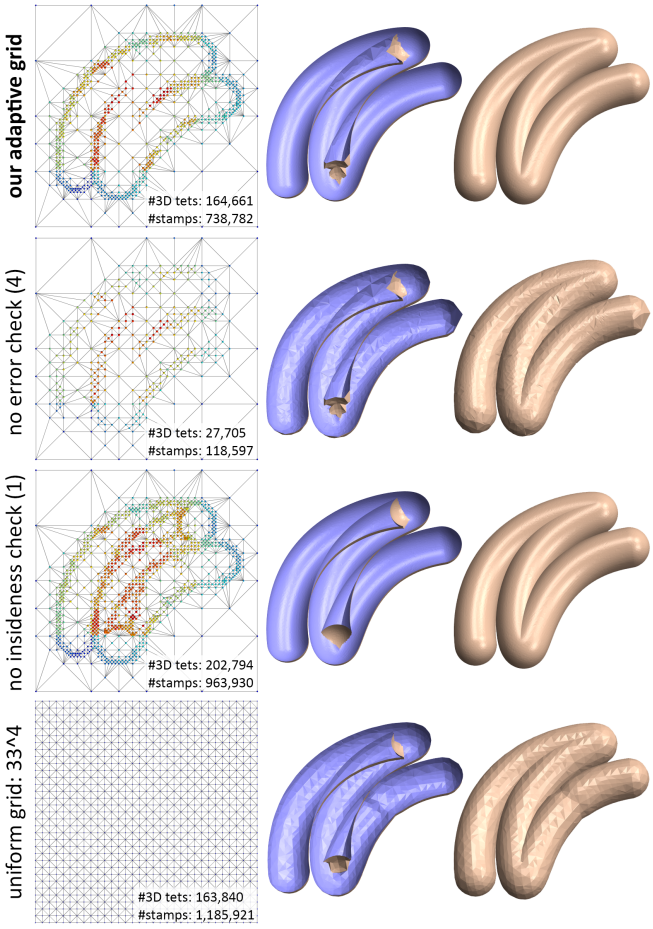


Fig. 16. Grid refinement using our method (top), without the approximation error check (2nd row), without the insideness check (3rd row), and using uniform refinement (bottom), showing the cross-sectional view of the refined grids (left) and the resulting envelopes (before computing the arrangement) in cut-away (middle) and full (right) views. On the left, grid points are colored by the number of time stamps (red means more).

(3) Can the silhouette set in the 4D column of s be well approximated by column-marching, up to some error threshold ϵ_{sil}? If not, s is refined spatially or temporally depending on the dimension that contributes the most error. Otherwise,
(4) Can the envelope in s be well approximated by polyhedron-marching and 3D projection, up to some error threshold ϵ_{env}? If not, s is refined spatially.

The first two checks determine if s contains a part of the sweep boundary, the subset of the envelope that does not lie inside the sweep volume. The last two checks ensure that the output of the surfacing algorithm - the discrete silhouette set and lifted envelope - in the column of s has sufficient accuracy. These checks are implemented using the zero-crossing tests and error bounds developed in [Ju et al. 2024] for simplices, utilizing a simplicial decomposition of the column of s. The implementation details are in Appendix B.

As an example, the refined grid for the rolling-ball sweep in Figure 6 is shown in a cross-sectional view in Figure 16 (top left). Note that the grid is more refined around the sweep boundary, both spatially and temporally (indicated by colors at the spatial grid points, which correspond to the number of time stamps). Grid points with more time stamps are located where different parts of the envelope come into close contact; this is where the silhouette set intersects more frequently with the timelines, and thus more temporal samples are needed (by our second check).

As an ablation experiment, we compare our method with several variations in Figure 16 (second to fourth rows). Removing the last check (on the envelope approximation error) leads to under-refinement around the envelope, and in turn a less smooth surface. On the other hand, removing the first check (on insideness with respect to the sweep volume) results in over-refinement around interior parts of the envelope that do not lie on the sweep boundary. Finally, a uniform grid yields a much coarser approximation of the envelope, even though the grid has a similar number of tetrahedra as our adaptively refined grid and more temporal samples.

8 Evaluation

We evaluated our method on a variety of 3D sweeps represented by implicit functions. Our method is implemented in C++. The only external package we used is the mesh arrangement program by [Zhou et al. 2016]. All experiments were conducted on a MacBook Pro with an M4 Pro chip and 48 GB of RAM. Code and data are available at https://github.com/Jurwen/Swept-Volume.

8.1 Parameters

Our grid generation algorithm is controlled by several parameters, including the initial spatial-temporal grid, the minimum edge length ϵ_{space} and time interval ϵ_{time} to decide when to stop processing the spatial and temporal queues, and the thresholds used in the refinement criteria (ϵ_{sil} and ϵ_{env}). We adopt a uniform tetrahedral mesh created by decomposing a 4^3 cubical grid using Kuhn's subdivision, where each grid point starts with 5 uniformly spaced time samples (and hence a total of 5^4 time samples). Surprisingly, this coarse initial grid is sufficient for all of our examples. Assuming the lifted sweep volume fits in a 4D unit cube, we fix $\epsilon_{\text{time}} = 2^{-7}$ and adapts ϵ_{space} to the accuracy goal by setting it to be $0.05 * \epsilon_{\text{env}}$.

The quality of the computed sweep boundary is mainly determined by ϵ_{sil} and ϵ_{env}, which respectively control the accuracy of approximating the silhouette set in $\mathbb{R}^4$ and the envelope in $\mathbb{R}^3$. To obtain more accurate results, both parameters need to be sufficiently small. A large ϵ_{sil} leads to a poor approximation of the silhouette set. This may cause the level set computed on the silhouette set to deviate significantly from the lifted envelope. Such deviation manifests as artifacts on the surface, which cannot be resolved by lowering ϵ_{env} and often lead to over-refinement, as highlighted in Figure 17 (top-right). On the other hand, a large ϵ_{env} produces a coarse approximation of the envelope, which cannot be improved by lowering ϵ_{sil}, as highlighted in Figure 17 (bottom-left). As smaller thresholds naturally lead to longer running times and larger output sizes, these two parameters may vary case-by-case depending on the need for accuracy versus speed. We use $\epsilon_{\text{sil}} \in [0.001, 0.005]$ and $\epsilon_{\text{env}} \in [0.0001, 0.0005]$ in our examples.

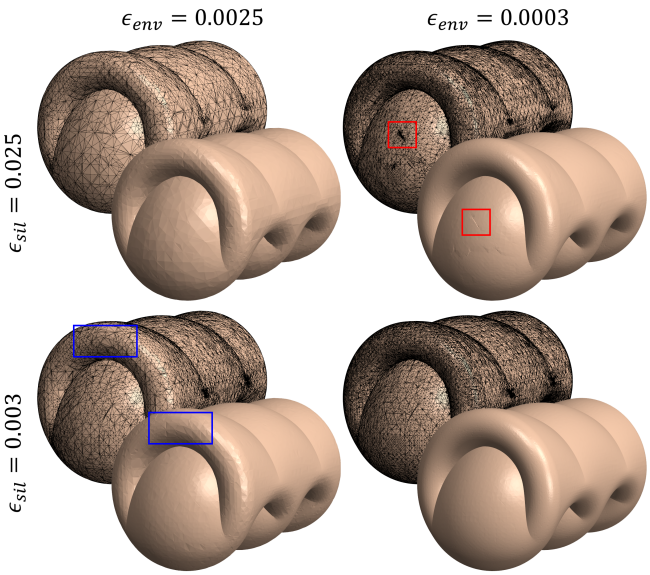


Fig. 17. Sweep boundaries (Figure 2) computed using different combinations of grid refinement parameters (ϵ_{sil} and ϵ_{env}). Boxes highlight over-refined (red) and under-refined (blue) areas.

8.2 Comparisons and results

We compare our method with the recent method of Sellan et al. [2021], which works on rigid sweeps (Equation 1) where the brush is defined by a mesh SDF (computed by libigl [Jacobson et al. 2018]). Sellan's method uses root-finding for generating smooth surfaces and adopts a continuation technique to efficiently trace the surface on a grid. However, as a volumetric method, it produces blurred creases and artifacts around closed-by surfaces, as shown in Figures 2 (c) and 6 (c). At a comparable output mesh complexity, our method produces sharp creases (see inserts in Figure 2 (b)) and can resolve extremely close surfaces (see inserts in Figure 6 (b)). Increasing the grid resolution in Sellan's method reduces, but does not eliminate, the rounded features and artifacts, while significantly increasing the output mesh size and running time (Figures 2 (d) and 6 (d)).

A key advantage of our method is that it applies to any generalized sweep where the brush may undergo morphological and even topological changes. To demonstrate the generality, we start with simple sweeps that morph between a torus and a ball, as in Figure 18, or between one and two balls, as in Figure 19. Observe that the computed sweep boundaries are rich with voids and sharp creases (e.g., the inserts in Figure 18 top). A more interesting morph, between a genus-5 cube and a genus-3 tetrahedron (both as quartic polynomials [Plantinga and Vegter 2007]), is shown in Figure 20. The sweep boundary has a complex interior structure, as seen in the transparent and cut-away views. Besides morphing, we can change the topology of the brush by offsetting its surface during sweeping. This is demonstrated in the spiraling and shrinking Fertility model in Figure 1, where the brush genus reduces from 4 to

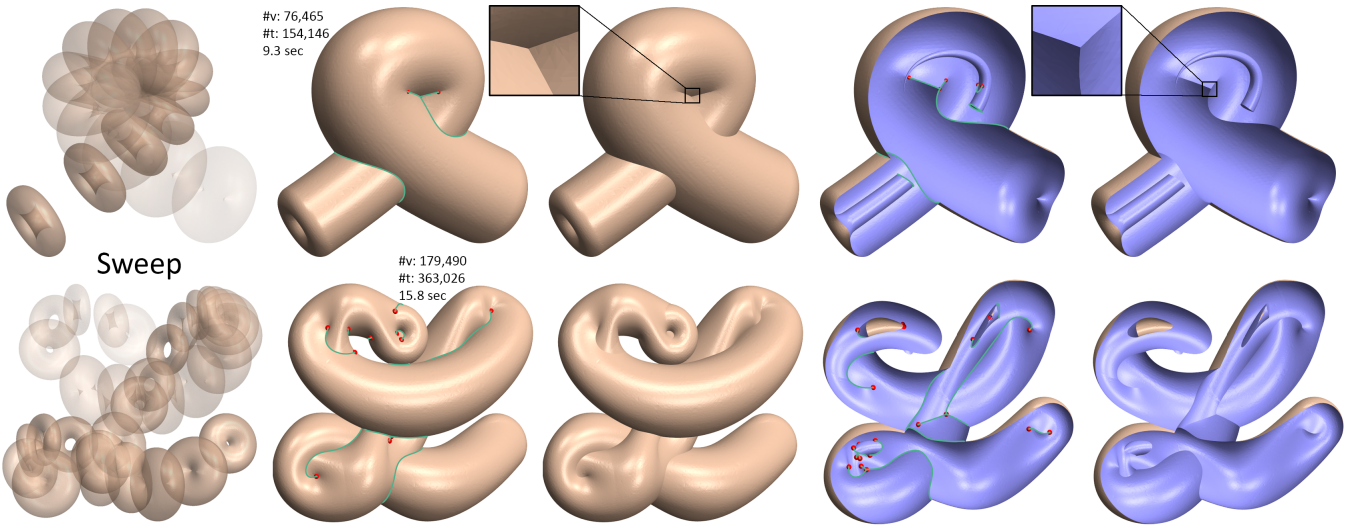


Fig. 18. Sweep boundaries of a torus morphing to a ball (top) and of a brush morphs back and forth between a torus and a ball (bottom) along spatial curves. The boundaries and cut-away views are shown both with and without marking the feature curves. The inserts zoom in on two sharp corners.

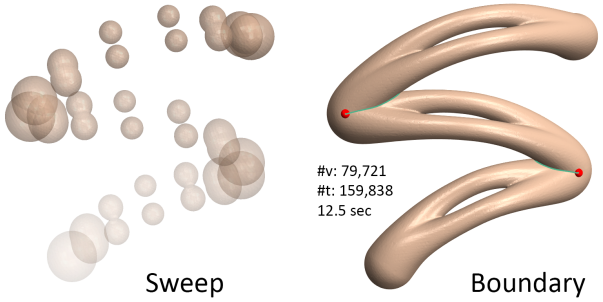


Fig. 19. A sweep that splits and merges.

0, and in the rolling and expanding wire ball in Figure 21, where the genus changes from 41 to 29. Observe that the computed sweep boundaries, equipped with sharp creases, exhibit complex geometry and topology (the rolling-ball sweep has 110 cavities).

Another example of a generalized sweep is the composition of multiple sweep functions using operators such as (soft) min and max, effectively performing boolean operations on their sweep volumes. We show a few sweeps defined in this way in Figure 22, inspired by a similar example in [Marschner et al. 2023]. Each sweep is defined by a soft-min (i.e., union) of two rigid sweeps (sweeping a sphere and a star). Observe that our method produces sharp intersections between the two rigid sweeps as creases, in addition to the sharp features on each component rigid sweep.

8.3 Performance

Our method takes between seconds to minutes for the examples shown in this paper (see notes in the figures). The runtime is dominated by grid generation, where the main factors are the time for each evaluation of the sweep function (and its gradient) and the geometric complexity of the sweep boundary. In addition, more stringent (lower) thresholds in the grid refinement criteria lead to more refinement and hence longer runtime. Figure 23 (left) plots the runtime of various steps of our algorithm as a function of the decreasing ϵ_{env} on the sweep in Figure 2. Observe that the timing for all steps exhibits a polynomial increase, which correlates with the growth in the number of geometric elements being generated, as plotted in Figure 23 (right).

9 Conclusion and discussions

We presented a new method for computing sweep volumes in 3D. Motivated by a novel characterization of the sweep boundary, our method combines both volumetric and mesh-based approaches to produce watertight, intersection-free surfaces that better approximate the geometric and topological features of the sweep boundary than existing methods. In addition, our method extends the scope of current methods to sweeps with changing shape and topology.

Our method has several limitations. First, our method does not guarantee to recover the correct topology of the sweep boundary, nor can we establish a geometric error bound. This is due in part to the top-down grid refinement algorithm, which may fail to sufficiently refine in some regions, for example, where the sweep function exhibits significant variations. Furthermore, our meshing algorithm may produce large errors near the singularities of the envelope (even though they are rare on the sweep boundary). Better grid refinement may be achieved by using higher-order local approximations in the refinement criteria, while explicitly modeling the envelope singularities could further improve the meshing accuracy. Second, computing the arrangement of the envelope surface explicitly as meshes can become time-consuming for complex sweeps. As the envelope is defined and computed implicitly in our method (as intersections of 4D level sets), utilizing implicit arrangement methods [Du et al. 2022] is a natural step towards improving scalability.

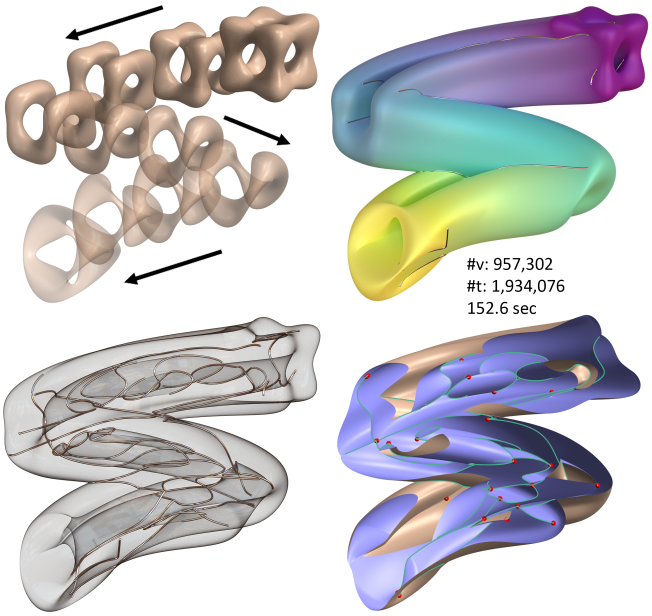


Fig. 20. Top-left: A genus-5 cube morphed into a genus-3 tetrahedron along the same zig-zag path as Figure 19. Rest: The time-colored sweep boundary with sharp creases (top-right), in transparency (bottom-left), and a cut-away view (bottom-right).

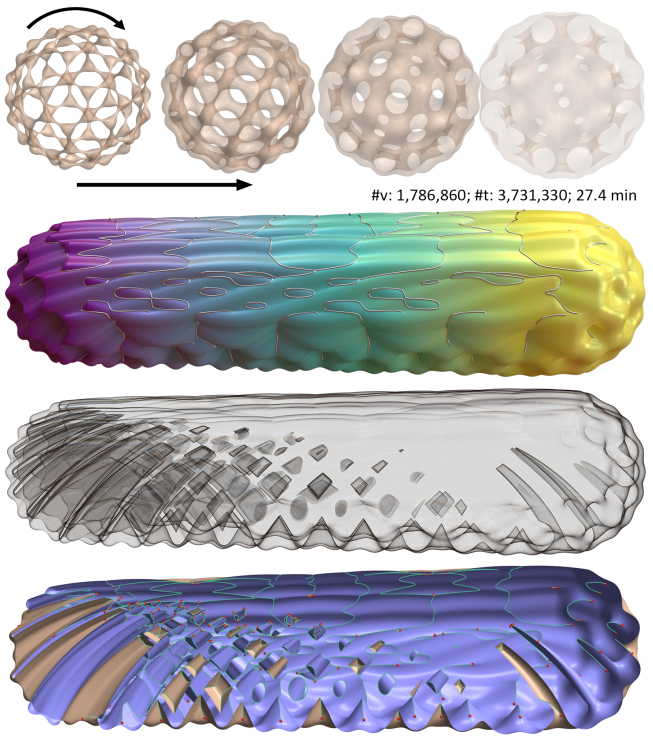


Fig. 21. Top: A wire-like ball rolls forward while offsetting, changing its genus from 41 to 29. Bottom: The time-colored sweep boundary and sharp creases (2nd row), in transparency (3rd row, creases hidden for clarity), and a cut-away view (bottom row).

Doing so will also maintain an implicit representation throughout our method, making our output ready for applications like CSG and offset. Finally, we will explore further generalization of our method, for example, to piecewise smooth implicit functions (e.g., a CSG shape with sharp edges) and sweeping in a multi-parameter domain (e.g., over a surface) instead of a single parameter (time).

Acknowledgments

This work is supported by NSF grant HCC-2401224 and a gift from Adobe Research.

References

Karim Abdel-Malek, Jingzhou Yang, and Denis Blackmore. 2001. On swept volume formulations: implicit surfaces. *Computer-Aided Design* 33, 1 (2001), 113–121.

Karim Abdel-Malek, Jingzhou Yang, Denis Blackmore, and Ken Joy. 2006. Swept volumes: fundation, perspectives, and applications. *International Journal of Shape Modeling* 12, 01 (2006), 87–127.

Karim Abdel-Malek and Harn-Jou Yeh. 1997. Geometric representation of the swept volume using Jacobian rank-deficiency conditions. *Computer-Aided Design* 29, 6 (1997), 457–468.

Steven Abrams and Peter K Allen. 1995. Swept volumes and their use in viewpoint computation in robot work-cells. In *Proceedings. IEEE International Symposium on Assembly and Task Planning*. IEEE, 188–193.

Bharat Adsul, Jinesh Machchhar, and Milind Sohoni. 2014. Local and global analysis of parametric solid sweeps. *Computer Aided Geometric Design* 31, 6 (2014), 294–316.

Bharat Adsul, Jinesh Machchhar, and Milind Sohoni. 2015. A computational framework for boundary representation of solid sweeps. *Computer-Aided Design and Applications* 12, 2 (2015), 181–191.

Eugene L. Allgower and Kurt Georg. 1989. Estimates for piecewise linear approximations of implicitly defined manifolds. *Applied Mathematics Letters* 2 (1989), 111–115. https://api.semanticscholar.org/CorpusID:123564143

Eugene L Allgower and Phillip H Schmidt. 1985. An algorithm for piecewise-linear approximation of an implicitly defined manifold. *SIAM journal on numerical analysis* 22, 2 (1985), 322–346.

Vladimir Igorevich Arnold, GS Wassermann, and RK Thomas. 1986. *Catastrophe theory*. Vol. 3. Springer.

John Baxter, Mohammad R Yousefi, Satomi Sugaya, Marco Morales, and Lydia Tapia. 2020. Deep prediction of swept volume geometries: Robots and resolutions. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 6665–6672.

Praveen Bhaniramka, Rephael Wenger, and Roger Crawfis. 2004. Isosurface construction in any dimension using convex hulls. *IEEE Transactions on Visualization and Computer Graphics* 10, 2 (2004), 130–141.

Denis Blackmore and Ming C Leu. 1992. Analysis of swept volume via Lie groups and differential equations. *The International Journal of Robotics Research* 11, 6 (1992), 516–537.

Denis Blackmore, Ming C Leu, and Frank Shih. 1994. Analysis and modelling of deformed swept volumes. *Computer-Aided Design* 26, 4 (1994), 315–326.

Denis Blackmore, Ming C Leu, and Liping P Wang. 1997. The sweep-envelope differential equation algorithm and its application to NC machining verification. *Computer-Aided Design* 29, 9 (1997), 629–637.

Denis Blackmore, Roman Samulyak, and Ming C Leu. 1999. Trimming swept volumes. *Computer-Aided Design* 31, 3 (1999), 215–223.

Jean-Daniel Boissonnat, Siargey Kachanovich, and Mathijs Wintraecken. 2023. Tracing Isomanifolds in d in Time Polynomial in d using Coxeter–Freudenthal–Kuhn Triangulations. *SIAM J. Comput.* 52, 2 (2023), 452–486.

Jean-Daniel Boissonnat and Mathijs Wintraecken. 2022. The Topological Correctness of PL Approximations of Isomanifolds. *Found. Comput. Math.* 22, 4 (aug 2022), 967–1012. https://doi.org/10.1007/s10208-021-09520-0

Marcel Campen and Leif Kobbelt. 2010. Polygonal boundary evaluation of minkowski sums and swept volumes. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1613–1622.

Antonio Castelo, Lucas Moutinho Bueno, and Marcio Gameiro. 2022. A combinatorial marching hypercubes algorithm. *Computers & Graphics* 102 (2022), 67–77.

Hao-Tien Lewis Chiang, Aleksandra Faust, Satomi Sugaya, and Lydia Tapia. 2020. Fast swept volume estimation with deep learning. In *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*.

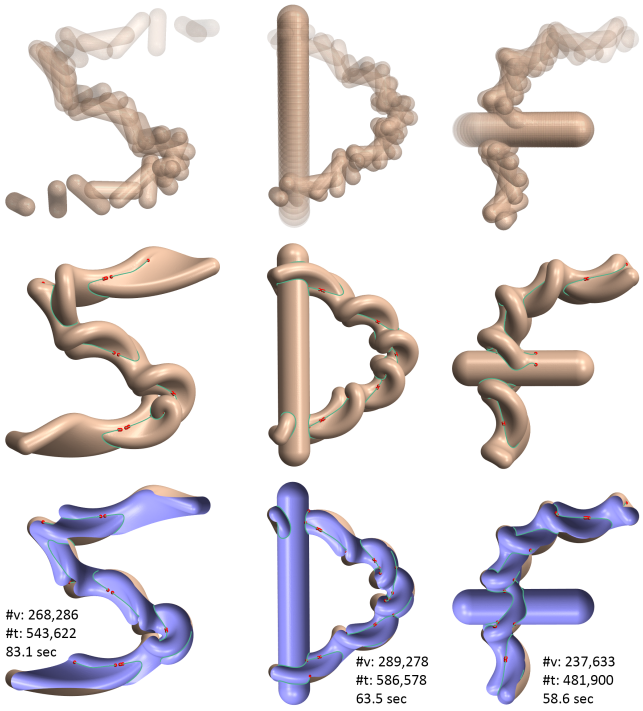


Fig. 22. Letter-like shapes created by soft-min (union) of two sweep functions (rigid sweep of a tube and a sphere), inspired by a similar example in [Marschner et al. 2023].

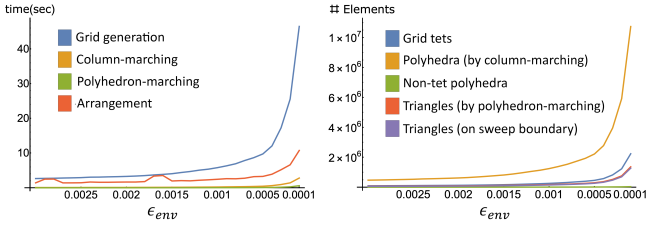


Fig. 23. Timing of each step (left) and the number of geometric elements (right) on the sweep in Figure 2 as ϵ_{env} decreases. An algebraic torus is used instead of a mesh SDF for faster evaluation.

Springer, 52–68.
Jason Conkey and Kenneth I Joy. 2000. Using isosurface methods for visualizing the envelope of a swept trivariate solid. In *Proceedings the Eighth Pacific Conference on Computer Graphics and Applications*. IEEE, 272–280.
Bruno Rodrigues De Araújo, Daniel S Lopes, Pauline Jepp, Joaquim A Jorge, and Brian Wyvill. 2015. A survey on implicit surface polygonization. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 1–39.
Sény Diatta, Guillaume Moroz, and Marc Pouget. 2019. Reliable Computation of the Singularities of the Projection in R3 of a Generic Surface of R4. In *MACIS 2019 - Mathematical Aspects of Computer and Information Sciences*. Gebze-Istanbul, Turkey. https://inria.hal.science/hal-02406758
Xingyi Du, Qingnan Zhou, Nathan A. Carr, and Tao Ju. 2022. Robust computation of implicit surface networks for piecewise linear functions. *ACM Transactions on Graphics (TOG)* 41 (2022), 1 – 16. https://api.semanticscholar.org/CorpusID:249917144
Laurent Dupont, Michael Hemmer, Sylvain Petitjean, and Elmar Schömer. 2007. Complete, exact and efficient implementation for computing the adjacency graph of an arrangement of quadrics. In *Proceedings of the 15th Annual European Conference on Algorithms* (Eilat, Israel) *(ESA'07)*. Springer-Verlag, Berlin, Heidelberg, 633–644.
Hüseyin Erdim and Horea T Ilieş. 2007. Detecting and quantifying envelope singularities in the plane. *Computer-Aided Design* 39, 10 (2007), 829–840.
Hüseyin Erdim and Horea T Ilieş. 2008. Classifying points for sweeping solids. *Computer-Aided Design* 40, 9 (2008), 987–998.
Hüseyin Erdim and Horea T Ilieş. 2010. A comparison of sampling strategies for computing general sweeps. *Computer-Aided Design* 42, 8 (2010), 657–669.
Jesse C Himmelstein, Etienne Ferre, and Jean-Paul Laumond. 2009. Swept volume approximation of polygon soups. *IEEE Transactions on Automation Science and Engineering* 7, 1 (2009), 177–183.
Horea T Ilies and Vadim Shapiro. 1999. The dual of sweep. *Computer-Aided Design* 31, 3 (1999), 185–201.
Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/.
Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 339–346.
Yiwen Ju, Xingyi Du, Qingnan Zhou, Nathan Carr, and Tao Ju. 2024. Adaptive grid generation for discretizing implicit complexes. *ACM Trans. Graph.* 43, 4, Article 82 (July 2024), 17 pages. https://doi.org/10.1145/3658215
J Kieffer and FL Litvin. 1991. Swept volume determination and interference detection for moving 3-D solids. (1991).
Young J Kim, Gokul Varadhan, Ming C Lin, and Dinesh Manocha. 2003. Fast swept volume approximation of complex polyhedral models. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*. 11–22.
François Labelle and Jonathan Richard Shewchuk. 2007. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph.* 26, 3 (jul 2007), 57–es. https://doi.org/10.1145/1276377.1276448
Chenxi Liu, Pierre Bénard, Aaron Hertzmann, and Shayan Hoshyari. 2023. Contesse: Accurate occluding contours for subdivision surfaces. *ACM Transactions on Graphics* 42, 1 (2023), 1–16.
Jinesh Machchhar, Denys Plakhotnik, and Gershon Elber. 2017. Precise algebraic-based swept volumes for arbitrary free-form shaped tools towards multi-axis CNC machining verification. *Computer-Aided Design* 90 (2017), 48–58.
Claudia Madrigal and Ken Joy. 1999. Generating the envelope of a swept trivariate solid. (1999).
Zoë Marschner, Silvia Sellán, Hsueh-Ti Derek Liu, and Alec Jacobson. 2023. Constructive solid geometry on neural signed distance fields. In *SIGGRAPH Asia 2023 conference papers*. 1–12.
Ralph R Martin and PC Stephenson. 1990. Sweeping of three-dimensional objects. *Computer-Aided Design* 22, 4 (1990), 223–234.
Chohong Min. 2003. Simplicial isosurfacing in arbitrary dimension and codimension. *J. Comput. Phys.* 190, 1 (2003), 295–310.
Bernard Mourrain, Jean-Pierre Técourt, and Monique Teillaud. 2005. On the computation of an arrangement of quadrics in 3d. *Computational Geometry* 30, 2 (2005), 145–164.
Enrique Outerelo et al. 2009. *Mapping degree theory*. Vol. 108. American Mathematical Soc.
Martin Peternell, Helmut Pottmann, Tibor Steiner, and Hongkai Zhao. 2005. Swept volumes. *Computer-Aided Design and Applications* 2, 5 (2005), 599–608.
Simon Plantinga and Gert Vegter. 2007. Isotopic meshing of implicit surfaces. *The Visual Computer* 23, 1 (2007), 45–58.
Angel Plaza and Maria-Cecilia Rivara. 2003. Mesh Refinement Based on the 8-Tetrahedra Longest- Edge Partition.. In *Proceedings of the 12th International Meshing Roundtable*. 67–78.
Helmut Pottmann and Martin Peternell. 2000. Envelopes-computational theory and applications. In *Spring conference on computer graphics*. 3–23.
Margot Rabl, Bert Jüttler, and Laureano Gonzalez-Vega. 2008. Exact envelope computation for moving surfaces with quadratic support functions. *Advances in Robot Kinematics: Analysis and Design* (2008), 283–290.
Lucas Martinelli Reia, Marcio Gameiro, Tomás Bueno Moraes Ribeiro, and Antonio Castelo. 2025. A fast high-dimensional continuation hypercubes algorithm. *Computers & Graphics* (2025), 104237.
María Cecilia Rivara. 1991. Local modification of meshes for adaptive and/or multigrid finite-element methods. *J. Comput. Appl. Math.* 36 (1991), 79–89. https://api.semanticscholar.org/CorpusID:120011902
Jarek Rossignac, Jay J Kim, SC Song, KC Suh, and CB Joung. 2007. Boundary of the volume swept by a free-form solid in screw motion. *Computer-Aided Design* 39, 9 (2007), 745–755.
Elmar Schömer and Nicola Wolpert. 2006. An exact and efficient approach for computing a cell in an arrangement of quadrics. *Computational Geometry* 33, 1-2 (2006), 65–97.
William J Schroeder, William E Lorensen, and Steve Linthicum. 1994. Implicit modeling of swept surfaces and volumes. In *Proceedings Visualization'94*. IEEE, 40–45.
Philipp Schwaha and René Heinzl. 2010. Marching simplices. In *AIP conference proceedings*, Vol. 1281. American Institute of Physics, 1651–1654.
Silvia Sellán, Noam Aigerman, and Alec Jacobson. 2021. Swept volumes via spacetime numerical continuation. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–11.

Alexei Sourin and A Pasko. 1995. Function representation for sweeping by a moving solid. In *Proceedings of the third ACM symposium on Solid modeling and applications*. 383–391.
Andreas von Dziegielewski, Rainer Erbes, and Elmar Schömer. 2010. Conservative swept volume boundary approximation. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*. 171–176.
Andreas Von Dziegielewski, Michael Hemmer, and Elmar Schömer. 2013. High precision conservative surface mesh generation for swept volumes. *IEEE Transactions on Automation Science and Engineering* 12, 1 (2013), 183–191.
WP Wang and KK Wang. 1986. Geometric modeling for swept volume of moving solids. *IEEE Computer graphics and Applications* 6, 12 (1986), 8–17.
Chris Weigle and David C Banks. 1996. Complex-valued contour meshing. In *Proceedings of Seventh Annual IEEE Visualization'96*. IEEE, 173–180.
Klaus Weinert, Shangjian Du, Patrick Damm, and Marc Stautner. 2004. Swept volume generation for the simulation of machining processes. *International Journal of Machine Tools and Manufacture* 44, 6 (2004), 617–628.
John D Weld and Ming C Leu. 1990. Geometric representation of swept volumes with application to polyhedral objects. *The International Journal of Robotics Research* 9, 5 (1990), 105–117.
Hassler Whitney. 1955. On singularities of mappings of Euclidean spaces. I. Mappings of the plane into the plane. In *Hassler Whitney Collected Papers*. Springer, 370–406.
Zhiqi Xu, Zhiyang Chen, Xiuzi Ye, and Sanyuan Zhang. 2007. Approximate the swept volume of revolutions along curved trajectories. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*. 309–314.
Jingzhou Yang and Karim Abdel-Malek. 2005. Approximate swept volumes of NURBS surfaces or solids. *Computer Aided Geometric Design* 22, 1 (2005), 1–26.
Xinyu Zhang, Young J Kim, and Dinesh Manocha. 2009. Reliable sweeps. In *2009 SIAM/ACM joint conference on geometric and physical modeling*. 373–378.
Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–15.

A Proof of Theorem 4.1

We consider a smooth sweep function $f(x,t) : \mathbb{R}^n \times [0,1] \to \mathbb{R}$ satisfying the genericity conditions in Section 4. We define a *layer* at a point $x \in \mathbb{R}^n$ as a continuous range of $t \in [0,1]$ such that $f(x,t) < 0$. We call the number of layers at x its *layer count*, and we are interested in characterizing points with layer count 0. The boundary of the set of all such points is the sweep boundary.

Our first observation is that, as x travels in $\mathbb{R}^n$, its layer count changes only when x crosses the envelope. Assuming that x avoids the singularities and self-intersections of the envelope, the layer count of x can only change at one of the following "critical" events:

(1) A layer appears or disappears at time 0.
(2) A layer appears or disappears at time 1.
(3) A layer appears or disappears at some time $t \in (0,1)$.
(4) Two layers merge, or one layer splits into two, at some time $t \in (0,1)$.

In the first event, $f(x,0) = 0$ but $f(x,\epsilon) > 0$ for any sufficiently small ϵ, implying that $f'(x,0) > 0$. As a result, $\{x,0\}$ is on the bottom cap. Similarly, in the second event, $\{x,1\}$ is on the top cap. In the last two events, the function $f_x(s) = f(x,s)$ has a critical point at $s = t$. In other words, $f'(x,t) = 0$. Since $f(x,t) = 0$, $\{x,t\}$ must be a contour point. In all events, x is the projection of some point on the lifted envelope, which is the union of the contour, bottom and top caps, and so x is on the envelope.

We next show that the exact change in the layers at these critical events depends on the direction of movement and the derivatives of f at the event location. Consider a regular envelope point x (i.e., away from self-intersections and singularities). Since there exists a unique time $t \in [0,1]$ such that $\{x,t\}$ is on the lifted envelope, only one of the critical events above happens at x. Let $\nabla_x f(x,t)$ be the first n components of the gradient ∇f at $\{x,t\}$. Note that $\nabla_x f(x,t)$ is normal to the envelope at x. Now, consider another point $y \in \mathbb{R}^n$ that moves past x in some direction v, such that $v \cdot \nabla_x f(x,t) < 0$. As illustrated in Figure 24, the change in the layers of y depends on t and, in some cases, the second-order time derivative f'':

(1) $t = 0$: a new layer appears at time 0.
(2) $t = 1$: a new layer appears at time 1.
(3) $0 < t < 1$ and $f''(x,t) > 0$: a new layer appears at time t.
(4) $0 < t < 1$ and $f''(x,t) < 0$: two layers merge at time t.

Observe that the layer count increments by 1 in the first three cases, but it decrements by 1 in the last case.

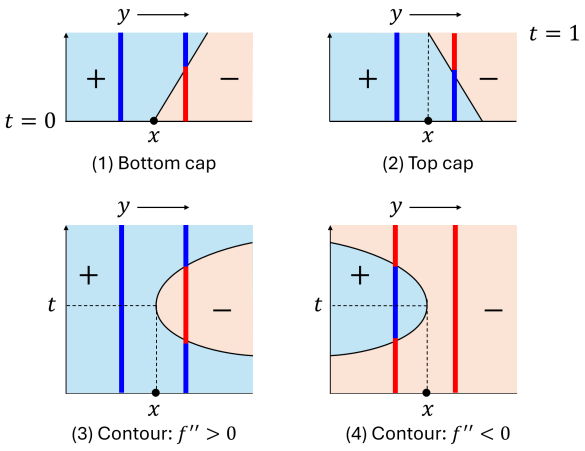


Fig. 24. Change in the layers (red vertical segments) of a moving point y at different critical events, as y moves past an envelope point x projected from a point on the bottom cap set (1), top cap set (2), or the contour set (3,4). Regions with positive (resp. negative) f are colored light blue (resp. red).

We will show that the outward orientation of the envelope defined in Section 4.2 has the following property: for any regular envelope point x projected from the lifted envelope point $\{x,t\}$, the outward normal at x is parallel to $-\nabla_x f(x,t)$ if $0 < t < 1$ and $f''(x,t) < 0$ (i.e., the last case above), and parallel to $\nabla_x f(x,t)$ otherwise. With this property, a point that moves across the regular part of the envelope from the "outside" to the "inside", as determined by the outward normal at the crossing point, always sees its layer count increased by 1.

To formalize this property, we denote by n_f, n_g the normalized gradient of f, g at $\{x,t\}$, and $\tau = \{\mathbf{0}, 1\}$ the unit vector along the positive time axis. Note that $n_g = \tau$ if $t = 0$ or 1, as $\{x,t\}$ lies on the bottom or top cap sets, and $n_g = \nabla f'(x,t)/|\nabla f'(x,t)|$ otherwise. As a result, $n_g \cdot \tau < 0$ if and only if $0 < t < 1$ and $\nabla f'(x,t) \cdot \tau = f''(x,t) < 0$. Let Σ be an orthonormal tangent basis of the lifted envelope at $\{x,t\}$, $\overline{\Sigma}$ be its projection, which is a tangent basis of the envelope at x, and $\overline{n_f}$ be the projection of n_f (by removing the last coordinate of n_f). Note that $\overline{n_f}$ is parallel to $\nabla_x f(x,t)$. Therefore, $\det(\{\overline{\Sigma}, \overline{n_f}\}) > 0$ (resp. < 0) implies that, in $\mathbb{R}^n$, the orientation represented by the tangent basis $\overline{\Sigma}$ is consistent with (resp. opposite to) the normal vector $\nabla_x f(x,t)$. We can now restate the property above as:

Proposition A.1. *If Σ is an outward tangent basis, then*

$$sign(\det(\{\overline{\Sigma}, \overline{n_f}\})) = sign(n_g \cdot \tau)$$

Proof. By the definition of outward basis in Section 4.2,

$$\det(\{\Sigma, n_f, n_g\}) > 0.$$

If $\{x, t\}$ lies on a cap set (i.e., $t = 0$ or 1), then $n_g = \tau = \{\mathbf{0}, 1\}$. Since the matrix $\{\overline{\Sigma}, \overline{n_f}\}$ is the upper-left block of the matrix $\{\Sigma, n_f, n_g\}$, and by the definition of determinants,

$$\det(\{\overline{\Sigma}, \overline{n_f}\}) = \det(\{\Sigma, n_f, n_g\}) > 0.$$

Thus the proposition holds, as $n_g \cdot \tau = 1$ is also positive.

Otherwise, $\{x, t\}$ lies on the contour set, where $f'(x, t) = 0$ and hence $n_f \cdot \tau = 0$. Let n_g^* be the projection of n_g onto the hyperplane orthogonal to n_f (i.e., the tangent plane of the zero-level set of f),

$$n_g^* = n_g - (n_f \cdot n_g) * n_f.$$

Observe that $n_g^* \cdot \tau = n_g \cdot \tau$. Since n_f and $(n_f \cdot n_g) * n_f$ are linearly dependent,

$$\det(\{\Sigma, n_f, n_g^*\}) = \det(\{\Sigma, n_f, n_g\}) > 0.$$

Let $n_g' = n_g^*/|n_g^*|$. Note that $n_g' \cdot \tau$ has the same sign as $n_g \cdot \tau$, and $\det(\{\Sigma, n_f, n_g'\}) > 0$. On the other hand, since n_g' is orthogonal to n_f, and both are orthogonal to Σ, which itself is an orthonormal basis, the vectors $\{\Sigma, n_f, n_g'\}$ form a positively oriented orthonormal basis of $\mathbb{R}^{n+1}$. We utilize the property that, for any orthonormal matrix $\begin{vmatrix} A & B \\ C & D \end{vmatrix}$ whose determinant is 1, $\det(A) = \det(D)$. In our case, $A = \{\overline{\Sigma}, \overline{n_f}\}$ and $D = n_g' \cdot \tau$. This implies

$$\operatorname{sign}(\det(\{\overline{\Sigma}, \overline{n_f}\})) = \operatorname{sign}(n_g' \cdot \tau) = \operatorname{sign}(n_g \cdot \tau),$$

which concludes the proof. □

Finally, we relate the layer count to the winding number of the outward-oriented envelope. The two numbers share several commonalities: both are integers; both are constant in any connected space away from the envelope; both increment or decrement by 1 as a point moves across the envelope, depending on the movement direction relative to the outward orientation of the envelope; and both are zero outside of the sweep volume. These commonalities imply that the two numbers have the same *absolute value* everywhere. As a result, points with zero layer count are exactly those with zero winding number. This concludes the proof of Theorem 4.1.

B Implementation of refinement criteria

We detail the implementation of the refinement criteria described in Section 7.2. We utilize the techniques developed in [Ju et al. 2024] for checking whether an n-simplex s intersects the zero-level set of a vector-valued function $F : \mathbb{R}^n \to \mathbb{R}^k$ for $k \geq 1$ (or equivalently, the intersection of the zero-level sets of k scalar functions), denoted by F^{-1}, and for bounding the error of approximating F^{-1} by linear interpolation. Specifically, a cubic Bezier simplex $\tilde{F}$ is constructed by sampling the values and gradients of F at the vertices of s. $\tilde{F}$ is represented by Bezier ordinates (each is a k-vector) at Bezier control points located on the vertices, edges, faces, and interior of s. Then s is considered to intersect F^{-1} if the k-dimensional convex hull spanned by the Bezier ordinates contains the origin. To bound the approximation error of of F^{-1} by the zero-level set of the barycentric interpolation of values of F at the vertices of s, denoted by $\overline{F}^{-1}$, the one-sided Hausdorff distance from $\tilde{F}^{-1}$ to $\overline{F}^{-1}$ is used. This distance is shown to be upper-bounded by a maximum error evaluated at all Bezier control points (Equation 14 in [Ju et al. 2024]).

To use these techniques, we decompose the 4D column of a tetrahedron s into 4-simplices. These simplices connect the time stamps at the vertices of s and are constructed using a simple and efficient routine. After sorting all time stamps in ascending order, we create the first 4-simplex from the first five time stamps in the sorted list L. These time stamps include one from each vertex (at time 0) and a fifth (positive) time stamp at some vertex. We then visit the remainder of L in order and, for each newly visited time stamp t, build a 4-simplex from t and the highest time stamp at each vertex that has already been visited. The simplicial decomposition is fast enough that it can be performed on-the-fly when checking a tetrahedron for refinement, thus avoiding the potentially large memory cost for storing these simplices.

We denote the set of 4-simplices constructed above as M. A tetrahedral face h of a 4-simplex $m \in M$ is said to be *horizontal* if h spans time samples from all four vertices of s. By construction, each 4-simplex has exactly two horizontal faces. We call the edge of m not part of either horizontal face the *vertical* edge of m. A vertical edge connects two time stamps at the same vertex.

Given the sweep and silhouette functions f, g, the four checks in Section 7.2 are implemented by testing zero-crossing and evaluating the error bound using the Bezier simplices of f, g inside the 4-simplices and, in some cases, inside the horizontal tetrahedra. To avoid sampling the 2nd-order derivatives of f (i.e., the gradient of g) at the vertices, we construct the Bezier simplex of g by degree-lifting the derivative of the Bezier simplex of f, which is quadratic, to a cubic Bezier simplex. The checks are implemented as follows:

(1) s is considered inside the sweep volume if f is negative on some horizontal face h of some 4-simplex $m \in M$. The latter is determined by slightly modifying the zero-crossing test in [Ju et al. 2024]: f is negative on h if all Bezier ordinates in the cubic Bezier simplex of h are negative.
(2) The column of s is considered to intersect the lifted envelope if any 4-simplex $m \in M$ intersects the intersection of the zero-level sets of f and g. We call such m a *candidate*.
(3) We check if the linear approximation error of the zero-level set of g in any candidate 4-simplex $m \in M$ is greater than ϵ_{sil}. If so, we compare the maximum error over the Bezier control points on the vertical edge of m with the smaller of the maximum errors over the Bezier control points on the two horizontal faces of m. If the former is greater, a temporal refinement is needed, and the corresponding time interval of the vertical edge is pushed into the temporal queue. Otherwise, we push s into the spatial queue for spatial refinement.
(4) We perform column-marching (Section 6.3) in the column of s and obtain a set of polyhedra approximating the silhouette set. We push s into the spatial queue for refinement if there exists some polyhedron p that intersects with some candidate 4-simplex, and either p is not a tetrahedron or the linear approximation error of the zero-level set of f in p exceeds ϵ_{env} after projecting to $\mathbb{R}^3$.