Swept Volumes via Spacetime Numerical Continuation

SILVIA SELLÁN, University of Toronto
NOAM AIGERMAN, Adobe Research
ALEC JACOBSON, University of Toronto and Adobe Research

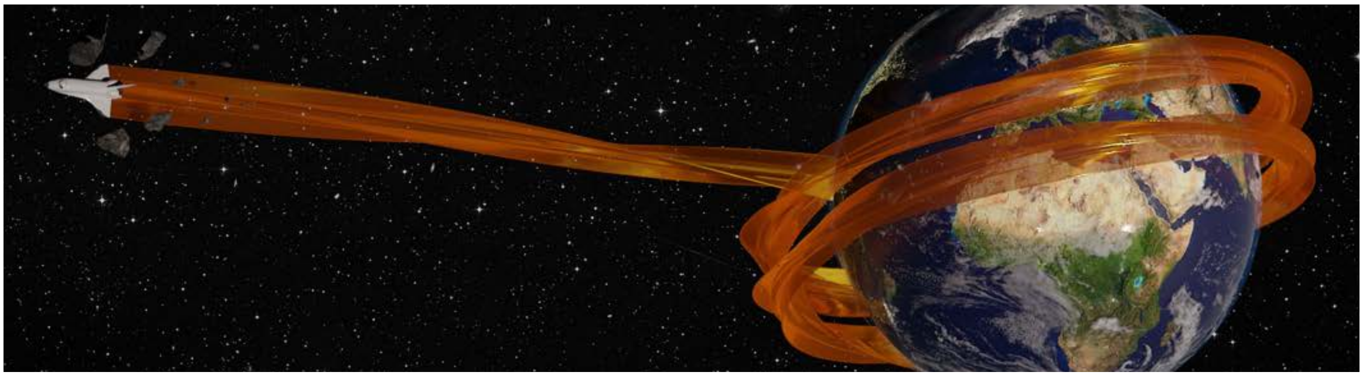


Fig. 1. We compute the surface of the volume (orange) *swept* by a moving space shuttle (top left). Instead of hunting everywhere in the cold, dark, vastness of space and time, our continuation algorithm traces along a 2D submanifold of a 3-manifold embedded in 4D, reducing the problem's dimension.

Given a solid 3D shape and a trajectory of it over time, we compute its *swept volume* – the union of all points contained within the shape at some moment in time. We consider the representation of the input and output as implicit functions, and lift the problem to 4D spacetime, where we show the problem gains a continuous structure which avoids expensive global searches. We exploit this structure via a continuation method which marches and reconstructs the zero level set of the swept volume, using the temporal dimension to avoid erroneous solutions. We show that, compared to other methods, our approach is not restricted to a limited class of shapes or trajectories, is extremely robust, and its asymptotic complexity is an order lower than standards used in the industry, enabling its use in applications such as modeling, constructive solid geometry, and path planning.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; Virtual reality; • **Mathematics of computing** → *Mathematical optimization.*

Additional Key Words and Phrases: swept volume, continuation algorithm.

ACM Reference Format:
Silvia Sellán, Noam Aigerman, and Alec Jacobson. 2021. Swept Volumes via Spacetime Numerical Continuation. *ACM Trans. Graph.* 40, 4, Article 55 (August 2021), 11 pages. https://doi.org/10.1145/3450626.3459780

Authors' addresses: Silvia Sellán, University of Toronto, 40 St. George Street, Toronto, M5S2E4, sgsellan@cs.toronto.edu; Noam Aigerman, Adobe Research, 601 Townsend St., San Francisco, 94103, aigerman@adobe.com; Alec Jacobson, University of Toronto and Adobe Research, 40 St. George Street, Toronto, M5S2E4, jacobson@cs.toronto.edu.

0730-0301/2021/8-ART55 $15.00
https://doi.org/10.1145/3450626.3459780

1 INTRODUCTION

A moving 3D object *sweeps* over space as a brush would sweep over a 2D canvas. The set of points that appear at some moment *inside* of the moving object constitute its *swept volume*. Hence, solid swept volumes lift the 2D brushstroke metaphor to the 3D-modeling setting (see Fig. 1), and we refer to the shape being swept as a *brush*. In 3D sculpting, the complement of the solid swept volume describes the removal of material by a moving chisel (see Fig. 2). Meanwhile, the swept volume of a robot or reconfigurable mechanism can be used to ensure safe clearance free of collisions.

Extracting a high-quality representation of the two-dimensional surface of a solid swept volume has proven to be an elusive problem. Nowadays, there are millions of polygonal meshes online with staggering detail, and modern mesh processing is mature for downstream tasks. It is particularly vexing to lack an extraction algorithm for an accurate mesh approximation of a moving mesh. Exact methods devolve into fragile and intractable surface meshing and Boolean operations for models found in the wild. The common response in practice (e.g., Adobe Medium) is to convert an input brush to an implicit representation (e.g., signed distance field) and then *stamp* the brush at discrete moments in time. The pesky choice of temporal resolution necessary for a smooth-looking output not only depends heavily on the complexity of the input brush *and* the input motion (see Fig. 3), but also frustratingly on the accuracy of the grid used for surface extraction. While simple to implement and parallelize, stamping suffers from performance complexity that scales with volume ($O(n^3)$) despite outputting a surface ($O(n^2)$).

In this paper, we consider the problem of extracting a high-quality mesh of the surface of the volume swept by an arbitrary input solid shape along an arbitrary trajectory. We propose a method which

Fig. 2. Subtracting the swept volume of complex brush shapes enables artistic control during 3D carving. 3D model by KellyBC under CC BY-SA 3.0.

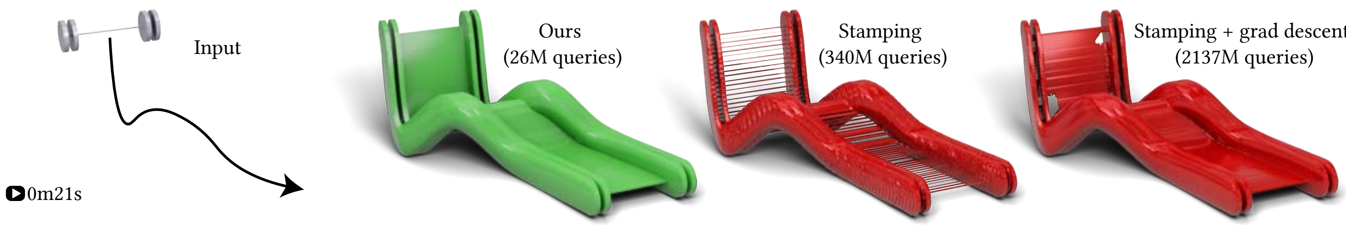


Fig. 3. Our method (green, center left) avoids the stroboscopic artifacts of the stamping at fixed time steps - the only previous work similar in generality (center right). Stamping *and then* doing gradient descent in time for each spatial point without our continuation algorithm (right) also presents artifacts despite incurring a significantly higher computational cost. *Scrub to the timestamp listed after any ▶ icon in our supplemental video for an animation.*

leverages the power of an intermediary implicit representation without inheriting the drawbacks of stamping. Our key insight is that each point on the surface of the swept volume has an associated "timestamp," corresponding to the moment in which the signed distance to the moving brush is minimized. This timestamp, viewed as a scalar field over $\mathbb{R}^3$, is piecewise continuous. We propose to "walk" along the two-dimensional surface of the swept-volume while tracking the small changes in timestamp value. By optimizing for the optimal timestamp (i.e., when the brush was closest) adaptively for each point, we ensure an alias-free output. Our approach can be understood as an application of the *method of numerical continuation.* Large swathes of 3D space are never even visited, resulting in an appropriately output-sensitive runtime (scaling with the sweep's surface complexity, generally $O(n^2)$).

Our method is extremely robust, and we have not encountered any failure case where it has produced an erronous result. Furthermore, our output surfaces consistently match the quality of methods specialized for specific classes of trajectories (see Figs. 4 & 5). Aside from its robustness, the power of our approach is in its generality, across a few respects: First, the input "3D brush" to our method could be any solid shape representation that admits a continuous implicit function, such as analytic signed distance functions, approximate signed distance functions (e.g., arising from constructive solid geometry operations or ShaderToy-esque metric manipulations), and robust winding-number [Jacobson et al. 2013] signed distances from triangle meshes and point clouds. Second, the input trajectory could be any representation of a rigid motion (beyond translations, screws and splines), but also encompasses articulated rigid bodies (effectively a union of each body's sweep) and Minkowski sums (generalizing 1D trajectory curve to a high-dimensional parametric space). Finally, the output of our basic method is the sweep's surface, retrieved via applying dual contouring [Ju et al. 2002] to the sparse set of voxels containing it. This avoids regions of space deep inside or outside the swept volume. This output sensitivity generalizes to directly contouring *interactions* with swept volumes such as constructive solid geometry operations (see Fig. 18). With the modern resurgence of implicit modeling (e.g., Adobe Medium, nTopology, Dream PS4, Claybook, Neural Implicits [Davies et al. 2021; Park et al. 2019]), our formulation via implicit functions affords flexibility not available with purely explicit methods.

We demonstrate the effectiveness and generality of our method through a variety of applications spanning 3D modeling, visual effects, and robotics clearance tasks. We further compare our method to the state of the art and report superior performance-over-accuracy ratio and surface quality.

2 WHY YET ANOTHER SWEPT VOLUME METHOD?

Computational methods for swept volumes are nearly as old as computer-aided design itself. Early work focused on accurately predicting the subtractive modeling processes of CNC milling [Sungurtekin and Voelcker 1986; Wang and Wang 1986]. Over the past decades, a wide variety of techniques for constructing swept volumes have appeared in the CAD, graphics, and robotics literature, with periodic surveys (e.g., [Abdel-Malek et al. 2006]). In this section, we describe how existing methods fall short for critical scenarios.

Since all points on the surface of a solid's sweep must originate from a point on the brush surface, it is natural to consider whether these points can be *explicitly* parameterized given a parametric or explicit representation of the brush surface. However tempting, exactly classifying a rigidly moving polyhedron (i.e., piecewise-flat) involves not just constructing ruled surface patches for each edge and face [Weld and Leu 1990], but also trimming their mutual intersections to remove components not contributing to the final surface [Blackmore et al. 1999]. Arrangements for planar meshes are already daunting, with very recent progress in robust algorithms relying on exact arithmetic [Zhou et al. 2016] or predicates [Cherchi et al. 2020].

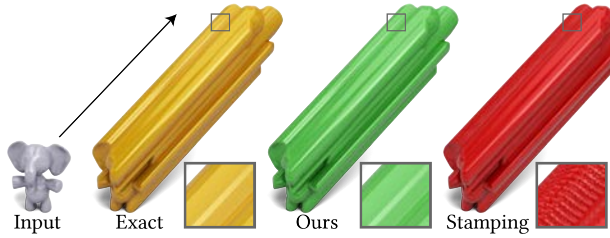


Fig. 4. Our method indistinguishably reproduces exact translational polyhedral sweeps, while stamping exhibits aliasing defects. 3D model by Ceh Jan under CC BY-SA 3.0.

These are not applicable to the ruled surfaces of a polyhedral sweep. Pure rational translations can be computed exactly [Zhou et al. 2016] (see Fig. 4) and screw motions can be well approximated with screw-specific analysis [Rossignac et al. 2007] (see Fig. 5), but these do not generalize to all motions or all classes of brushes. The situation worsens when compositing sweeping operations with other solid operations such as Booleans or offsetting [Pavic and Kobbelt 2008]. One option is to approximate surface patches after each operation with point clouds [Peternell et al. 2005], triangle meshes [Abrams and Allen 2000], or distance fields [Kim et al. 2004; Zhang et al. 2009], relying on meshing, *ad hoc* flood filling or contouring to assemble a final output surface mesh. These explicit methods *accumulate* error in a hard to control manner.

Campen and Kobbelt [2010] approximate the surface of swept volumes of polyhedra by first discretizing the input motion as piecewise-linear vertex displacements and then generating a superset of candidates from this motion. These must be trimmed and stitched to form the output. Intersections must be conducted robustly to ensure a watertight output mesh. Special purpose culling rules must be used to avoid the intractable problem of handling all possible intersections. Although code is not available for a direct comparison, we are not confident that this method will be robust in the presence of many intersections, such as in our Fig. 8. Furthermore, our algorithm accepts any input rigid motion without approximating it, we are not limited to polyhedral inputs and we remove the need for an elaborate post-facto cleanup.

It should be noted that for moving polyhedra, if we insist on a polygonal mesh as the eventual output, then by design we have to accept some approximation error. We should not care whether that error comes from attempting to triangulate exact surface-patches, or contouring an implicit function. We *do* care that the error is controllable (e.g., by choosing the resolution of the underlying representation), creases are well approximated, and the construction is robust and efficient.

Compared to explicit representations, a sweeping implicit is simply stated mathematically. If the input brush solid is the set of points $\mathbf{x} \in \mathbb{R}^3$ such that some continuous function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ is negative ($f(\mathbf{x}) < 0$), then the swept volume along some time-parameterized rigid motion $\mathbf{T} : [0, 1] \longrightarrow SE(3)$ is represented as a new implicit function taking the *minimum* of the brush's implicit evaluated relative to the motion over time:

$$f^\star(\mathbf{x}) = \min_{t \in [0,1]} f\left(\mathbf{T}(t)^{-1}\mathbf{x}\right). \tag{1}$$

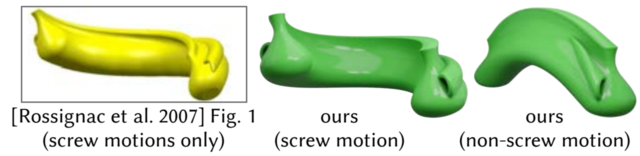


Fig. 5. Our method replicates the main result of Rossignac et al. [2007], while generalizing beyond their restricted class of motions.

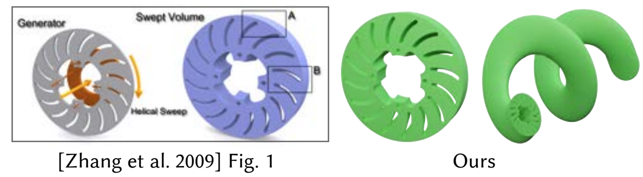


Fig. 6. Our results are visually identical to those of Zhang et al. [2009], but our method is more general and can work on sweeps theirs cannot.

Specifically, if f is a signed distance function then $f^\star$ will be an upper bound on the signed distance to the sweep (exact outside, underestimate inside, cf. [Quílez 2020]).

Since the input and output are both implicits, swept volumes slide neatly into the larger implicit modeling and rendering frameworks [Wyvill et al. 1986]. However, the simplicity of Eq. (1) crumbles under scrutiny when turning to implementation. For special combinations of brush functions (e.g., compositions of a small number of analytic functions) and motion parameterizations (e.g., polynomial splines) root finding can be employed to solve Eq. (1) [Schmidt and Wyvill 2005; Sourin and Pasko 1995]. In contrast, the "R-functions" used by Sourin and Pasko [1995] avoid minimization during aggregation. While simplifying some derivations, their analogous $f^\star$ has distorted values away from the zero level-set, precluding direct extraction of positive offset surfaces, especially useful in carving or robotic clearance problems. Even within their restricted settings, numerical root finding is employed at each evaluation point with some aspect of global search to avoid local minima. Lacking our continuation-based method, they scale poorly to general inputs and local minima leave outputs riddled with defects (see Fig. 3).

One way to sidestep the pitfalls of numerical root-finding is not to do it. Instead, one can march at constant time steps, and for each time step fill-in all implicit values at that time step (e.g., on a grid), a method we refer to as *stamping*. Stamping lies in direct comparison to us, as it is compeletely general (f must only be defined at any point $\mathbf{x}$ and moment in time t), and in fact applicable to a larger set of scenarios (e.g., non-rigid deformations). However, we demonstrate superior surface quality and performance trend. Indeed, combined with grids of precomputed f values and adaptive background grids, stamping can be streamlined for memory efficiency [Von Dziegielewski et al. 2012], but ultimately scales poorly with volume, $O(n^3)$ [Garg et al. 2016; Menon et al. 1994; Schroeder et al. 1994]. Furthermore, the choice of the sampling rate is non-trivial: Sampling too densely in time *fills in* values deep inside the swept volume as the surface of the brush marches forward. Sampling too sparsely leads to temporal aliasing becoming visible (see Fig. 3). This issue is aggravated by the fact that different points move at different speeds (e.g., through a rotation), hence some spatial regions require

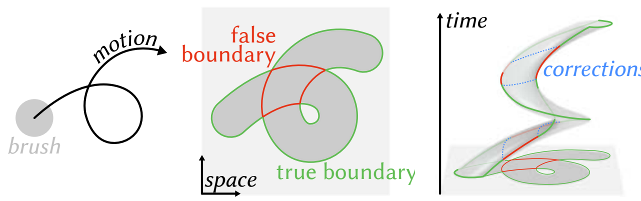


Fig. 7. Given an input brush and motion, characteristic points (cf. [Peternell et al. 2005]) trace *true* boundaries and *false* boundaries of the swept volume. Previous explicit methods pose surface extraction of the swept volume as a spatial arrangement problem. Instead, we conceptually consider the space-time hypersurface (codimension one) and trace continuously along the submanifold (codimension two). Any accidentally explored false boundaries are corrected when approached from spatially equivalent points.

finer time sampling than others. Frustratingly, the speed of stamping necessary to get an alias-free surface after contouring (e.g., via [Lorensen and Cline 1987]) depends on the choice of background grid resolution. Thus, the number of time samples k in the runtime complexity $O(kn^3)$ is effectively dependent on the grid resolution n (observationally roughly $O(n^4)$).

Contouring static implicit functions invites a similar discussion of volumetric (e.g., [Lorensen and Cline 1987]) versus output-sensitive complexity (e.g., [Bloomenthal 1988; Wyvill et al. 1986]). Bloomenthal casts the tracing of the output surface in the context of the *method of numerical continuation* (see, e.g., [Allgower and Georg 2003]). The core idea being that once a point $\mathbf{x}$ is found such that $f(\mathbf{x}) = 0$ the necessarily continuous surface must cross grid cells neighboring $\mathbf{x}$. Checking only neighbors of previously identified surface points avoids visiting the full volume of space. Our method is in essence a continuation method, applied to the minimization problem (Eq. (1)). However, we apply the continuation not to the minimum itself, but to its corresponding argmin:

$$t^\star(\mathbf{x}) = \underset{t \in [0,1]}{\operatorname{argmin}} \; f\left(\mathbf{T}(t)^{-1}\mathbf{x}\right). \tag{2}$$

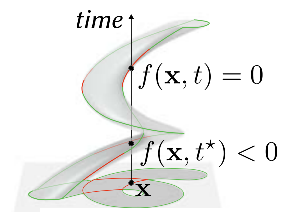
This stems from our core observation: $t^\star$ is piecewise-continuous over space, enabling our continuation method to propagate argmin values, in turn resulting in a robust and efficient algorithm.

3 SUBMANIFOLDS IN SPACETIME

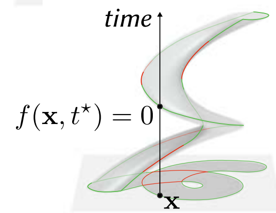
For simplicitiy, let us discuss swept volumes over a 2D example, where visualization is simpler. In Fig. 7, we show a prototypical self-intersecting motion of a solid brush creating a swept 2D "volume". Turning the plane sideways and adding a new dimension for time, we visualize the motion's *spacetime* surface. If we imagine shining a light from above, then the swept volume is the solid shadow captured on the spatial plane. The occluding contours and silhouettes represent false and true boundary curves of the swept volume, respectively. Each curve is composed of continuous parts of the corresponding spacetime curves.

Our goal will be to walk along just the low-dimensional submanifold of curves on the spacetime surface and output a continuous discretization of the true swept-volume boundaries.

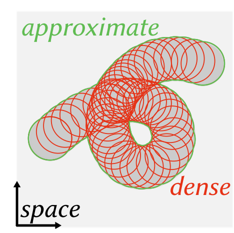
If our brush can be represented with an implicit function f, then for any point $\mathbf{x}$ in the dark region in Fig. 7 corresponding to the swept volume there exists some time t such that $f(\mathbf{x}, t) < 0$ (by slight abuse of notation, we define $f(\mathbf{x}, t) := f\left(\mathbf{T}(t)^{-1}\mathbf{x}\right)$). The true and false boundaries are *both* places where there exists a time t such that $f(\mathbf{x}, t) = 0$. Further, at these moments in time necessary optimality conditions will hold: $\partial f / \partial t = 0$. However, only for true boundaries will t be the *global* minimizer (i.e., $t = t^\star(\mathbf{x})$, see Eq. (2)).



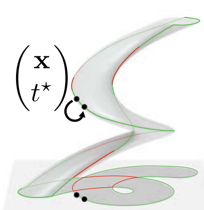
For example, consider a point $\mathbf{x}$ on a red false boundary, corresponding to the time t when $f(\mathbf{x}, t) = 0$. Since this point also lies deep in the dark ($f < 0$) swept volume, there must exist some other moment in time $t^\star$ where f reaches its minimal (negative) value: $f(\mathbf{x}, t^\star) < 0$.



A point $\mathbf{x}$ on the true boundary curve will reach $f = 0$ at its globally optimal time $t^\star$. There will *not* exist any other moment in time where f becomes negative. Our method seeks to identify all true boundaries by continuously walking in *spacetime* while avoiding overexploring false boundaries. These contours are one dimension less than the full swept volume, so our prospect for performance savings is high.



In contrast, naively stamping at discrete moments in time can be understood as a poor approximation of *both* the true and false boundaries. Stamping ignores the continuity and low dimensionality of the spacetime entities and proceeds in the spatial domain. Even ignoring that approximate true boundaries are aliased revealing the discretization, stamping litters the interior of the swept volume with candidate false boundaries. Each stamp contributes new false boundaries that must be trimmed away by subsequent stamps. As these become dense, performance suffers asymptotically.



Our novel idea is to conduct a self-correcting contouring of the *codimension two* manifold pre-image of the swept volume boundary in $d + 1$ *spacetime* rather than directly in d-dimensional space. Inductively, assuming we have already identified some point $(\mathbf{x}, t^\star)$ lying on this manifold, we conduct a discrete step-and-project in both space and time to identify a neighboring point also lying on the manifold. Once all points are determined, we may simply throw away the auxiliary time values. Conducting the step-and-project operation requires care, but we will see that working in spacetime greatly simplifies our algorithm.

With our crucially different understanding in place, we may now describe how we efficiently conduct contouring in spacetime ensuring a geometrically and topologically valid output. Our two-dimensional picture holds analogously when we lift the problem

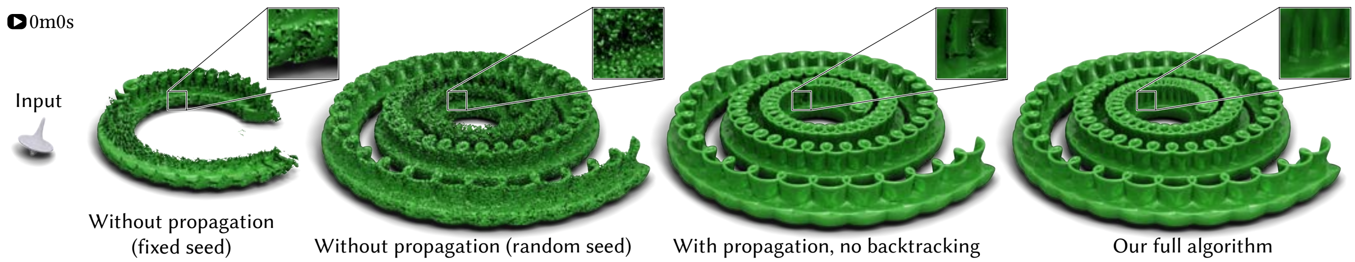


Fig. 8. An ablation study of our algorithm shows the deterioration in results by omitting the propagation of $t^\star$ and using fixed (left) or random (middle) time seeds, or omitting the correction steps in our algorithm (second to right). Our method (right) recovers the correct surface.

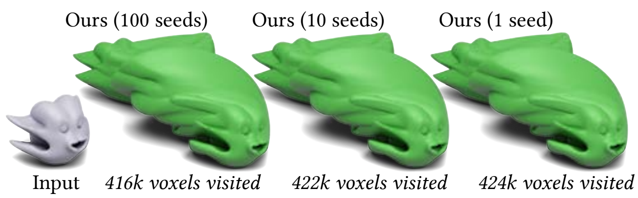


Fig. 9. Our algorithm is robust to the number of initial seeds in our queue, even in a sweep with self-intersections. 3D model by Jenna Stoeber under CC BY-NC-SA 4.0.

to moving solid brushes in 3D, so long as we carefully track codimensionality. Spacetime is now 4D, and our moving brush extrudes a hypersurface (3-manifold in 4D spacetime). Its cast shadow is again 3D (codimension zero) caught in space. Rather than walk along curves in spacetime, we will grow outward along the two-dimensional (codimension two) submanifold spidering around the spacetime hypersurface.

4 NUMERICAL CONTINUATION METHOD

We assume we receive as input an implicit representation f of the solid 3D brush (in Section 5, we demonstrate how working with implicits as input seamlessly enables many other input representations) and a function $\mathbf{T}(t)$ describing the rigid motion of the brush over a unit duration of some fictitious time. We next describe and discuss the continuation method we use to extract the 2-manifold embedded in 4D spacetime consisting of the spatial swept volume's surface coupled with the extra temporal coordinate $t^\star$.

4.1 Voxel grid representation

Our raw output is an implicit representation of the swept volume's surface, represented via a sparse voxel grid in 3D. The main parameter of our method is the grid resolution, i.e., the spacing between points on the grid, $h \in \mathbb{R}^+$. Internally, we also store a 4^{th} coordinate representing $t^\star$, denoted $t_i^\star$ for vertex $\mathbf{x}_i$, to distinguish it from the unknown ground truth $t^\star(\mathbf{x}_i)$. Similarly, we denote the value of the implicit for the vertex as $f_i^\star$.

Both the voxels and the vertices of the voxel-grid play a role in our computation. On one hand, the vertices store the computed values (i.e., $t_i^\star, f_i^\star$) and can be thought of as *semidiscrete* objects in 4D. Their location in space is discretized by the step size h (and fixed), while they are endowed with a continuous time value (which we modify via optimization). On the other hand, the voxels are used as the "implicit volume element" - we query them and evaluate each voxel's 8 vertices' values to infer whether the zero levelset (the surface) passes through it or not, and to decide whether to progress to its neighboring voxels.

We determine whether the surface passes through a voxel by checking whether $f_i^\star$ changes sign over the voxel's 8 vertices – since we only sample grid points, vertices rarely if ever land exactly on the surface of the swept volume: i.e., in general $f\left(\mathbf{x}_i, t_i^\star\right) \neq 0$. This is not a problem as we merely need to track for which *edges* $\{\mathbf{x}_i, \mathbf{x}_j\}$ of the grid does $f_i^\star$ change sign: by continuity, there must exist some $\mathbf{x} \in \{\mathbf{x}_i, \mathbf{x}_j\}$ for which $f\left(\mathbf{x}, t^\star(\mathbf{x})\right) = 0$ [Lorensen and Cline 1987]. We only include voxels that exhibit this sign change in our output, making it immediately digestible by off-the-shelf sparse contouring methods (e.g., [Bloomenthal 1988; Ju et al. 2002]).

4.2 Method

Our method is in essence a region-growing method reconstructing the 2D surface in 4D space. Each probe outward from the frontier of the reconstructed surface consists of a discretized spatial step (moving on the grid to a neighboring voxel) and a continuous optimization for the temporal coordinate, to project it back to $t^\star$. Additional rules stop the region growing at points straying away from the reconstructed surface, and backtracking to correct wrongful assignments (a minimum is discovered to be local rather than global). Fig. 8 shows an ablation study of our algorithm.

For region growing, we keep a (non-priority) queue of voxels to visit. Each voxel v on the queue also holds an initial guess of its temporal component t_v (to be used as initialization in the optimization of the voxel's vertices' values). We visit voxels as we pop them from the queue. We begin by assuming we have a small number of *seed* locations $(\mathbf{x}, t^\star)$ known to be on the surface of the swept volume (i.e., $f(\mathbf{x}, t^\star) = 0$).

We initialize our queue with the voxels containing the identified seed locations *lifted* to spacetime with corresponding seed's $t^\star$. When popping a voxel v with time value t_v from the queue, we visit each of the 8 voxel vertices i to compute its associated $t_i^\star$, and compare it to any previously-computed value it stores. Consider a vertex with spatial position $\mathbf{x}_i$. Since we wish to keep it *fixed* to the voxel grid, we move the only degree of freedom available — the temporal component — in order to bring the point as close as possible to the codimension one hypersurface in 4D. We locally

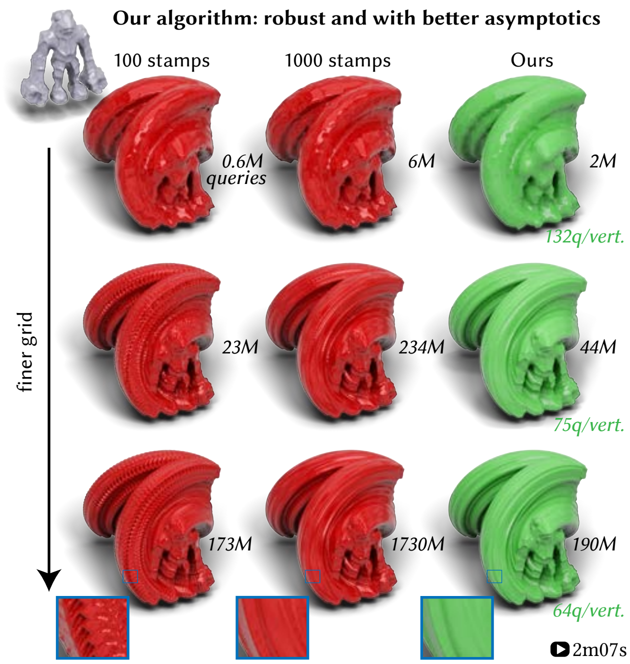


Fig. 10. For stamping, the number of fixed-time stamps required for a good result is tied to the spatial resolution: while few stamps may look good for coarse spatial resolutions, more and more stamps are required the finer the resolution gets. Our method doesn't present this undesirable behaviour. 3D model by Perry Engel under CC BY-NC 4.0.

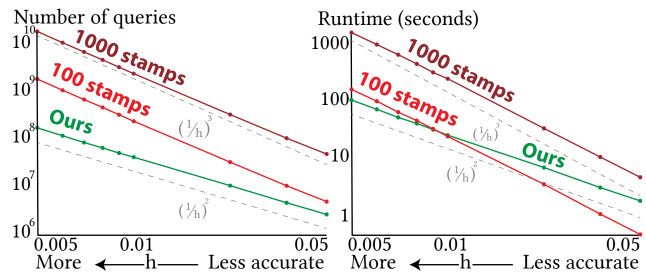


Fig. 11. Our method demonstrates superior performance over stamping, typified by results in Fig. 10 using a background grid for implicit querying.

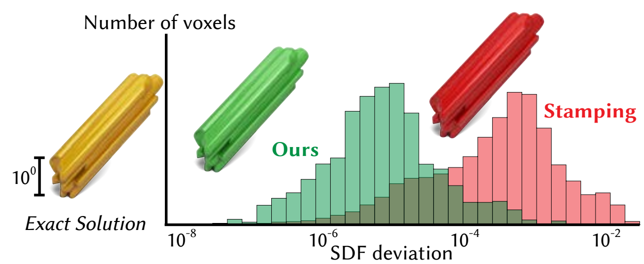


Fig. 12. Our algorithm's produced SDF values are several orders of magnitude more accurate than a similar runtime stamping's.

solve this 1D projection problem by conducting gradient descent to minimize $f(\mathbf{x}, t)$ starting at t_v. We simply employ a backtracking line search [Boyd and Vandenberghe 2004]. Let the time value (the argmin) resulting from gradient descent for this voxel corner be t_i. If this corner has never been visited before, then we set $t_i^\star \leftarrow t_i$ and $f_i^\star \leftarrow f(\mathbf{x}_i, t_i)$. Otherwise, we've visited this corner before and need to see whether we have a *clash* in our hypothesis for $t^\star$, which we resolve as follows.

If the new implicit value at $\mathbf{x}_i$ is smaller than the older one, $f(\mathbf{x}_i, t_i) < f_i^\star$, then we've identified a *correction*, meaning the previous value stored was of a strictly-local minimum, and our new value is the new candidate for the global minimum. Hence, we set $t_i^\star \leftarrow t_i$ and $f_i^\star \leftarrow f(\mathbf{x}_i, t_i)$ and add all other voxels incident on this corner to the queue endowed with time value t_i. Adding the voxels back to the queue initiates another front propagation that will recursively reevaluate any corner that took part in the front stemming from the wrongful local minimum. If the opposite condition holds, i.e., the old implicit value is smaller than the new one, $f_i^\star < f(\mathbf{x}_i, t_i)$, then we are *currently* tracing a local minimum, so we re-add the current voxel v to the queue with the cached $t_i^\star$, to have its other corners recomputed using this time value.

Lastly, if any corner of a voxel was updated, then we consider each edge $\{i, j\}$ of the voxel for which the signs of $f_i^\star$ and $f_j^\star$ differ. All other voxels neighboring this edge are added to the queue twice (once with $t_i^\star$ and once with $t_j^\star$). We restate the algorithm above in pseudocode in the Appendix B.

Interval basin caching. Voxels and grid vertices may be visited many times during the queue processing. To avoid re-optimizing the same time values over and over again for the same vertex, we take advantage of the 1D nature of the $t^\star$ optimziation.

Consider that a vertex has previously started a descent with value t_0 and converged to the nearest local minimizer t_1. If we visit this vertex again and request to start a descent with a value t_2 that lies in the interval spanned by t_0 and t_1, then we can skip the numerical descent and return t_1 immediately as t_2 must lie along the way down from t_0 to t_1. In general, a vertex may collect many temporal intervals. We may merge these using an interval tree as we progress, keeping a map from interval "basins" to associated local minima.

Extracting initial seeds. Our continuation algorithm needs a few initial "seeds": spacetime points $(\mathbf{x}, t^\star)$ such that $f(\mathbf{x}, t^\star) = 0$. Similarly to Peternell et al. [2005], we look for points whose velocity and normal vector are orthogonal, a necessary condition for lying on the sweep's surface. We sample time at coarse regular intervals (we use 10 intervals for all examples). For each time t_i, we draw 100 random points on the brush. For each point, if the orthogonality condition is satisfied up to some tolerance (i.e., dot product less than 0.01), we add its corresponding voxel to the queue endowed with time t_i. Since we are sampling from a superset, we conservatively compute many of these points (in our examples, 100) and start the continuation from all of them. While necessary but not sufficient, we have never encountered an example in which we failed to recover the correct swept volume surface, even when significantly reducing the number of initial seeds (see Fig. 9). If such example exists, we conjecture seeding more aggressively resolves it.

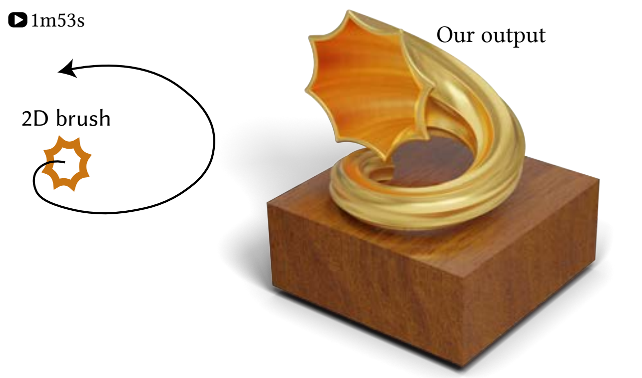


Fig. 13. Our method can robustly sweep thin, 2D curves, by sweeping a small offset from them.

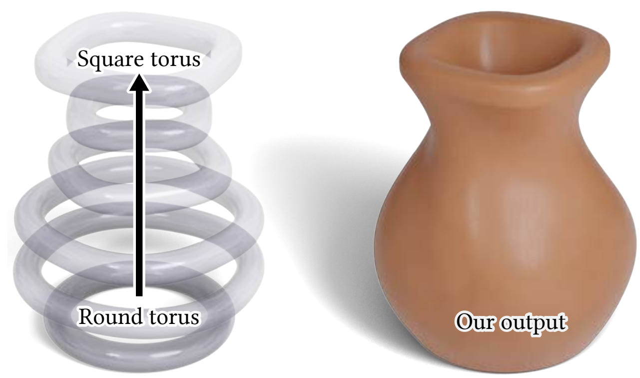


Fig. 14. Our method effortlessly sweeps a torus changing shape over time.

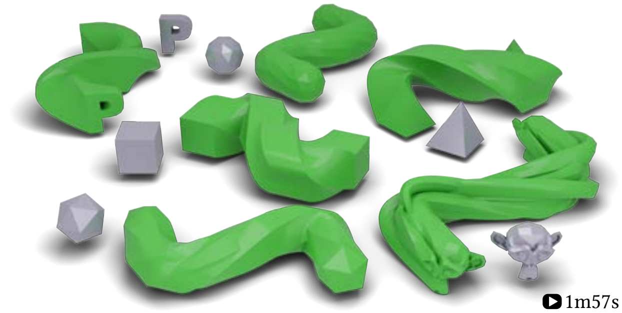


Fig. 15. Different geometric brushes produce different artistic profiles.

4.3 Discussion

Our method is guaranteed to terminate (queue only grows when new minima are found and they are finite for a non-fractal f), and also guaranteed to output voxel data ensuring a contouring of a closed surface (no crossing edges may appear on the sparse-voxel boundary). This is little comfort as the same is true of stamping.

The major difference for our method is that each update of a grid vertex is a small finite step from its neighbor's previously computed optimal value, leveraging the piecewise continuity of $t^{\star}$. This is what enables us to perform significantly fewer queries than stamping. In fact, in Appendix A we show that under sufficient regularity, and assuming that the implicit function is a signed distance field (as it is in most examples in this paper), then given a desired accuracy tolerance ε of the output, our method requires a number of queries which is sublinear, $O(\log(1/\varepsilon))$, while stamping requires a number of queries which is linear, $O(1/\varepsilon)$.

While it is possible to step *over* a globally optimal value and into a nearby local minimum, we do not often observe this. Even if this does happen once, all other neighbors still have an opportunity to send an improvement.

Theoretically, a catastrophic sequence of missed global optimums could lead to entire patches missing in the output (false positives) or erroneously retained (false negatives). With even mild initial seeding, we have never witnessed this in all of our experiments.

Compared to stamping, whose worst case behavior spans the entire discretized swept *volume*, in the worst case, our method traces all false boundaries, only to correct them later. We similarly do not witness this. In our experiments false boundaries are briefly explored but just as quickly corrected. If even a single point on a false boundary component is corrected then the queue acts as a fast breadth first correction for the whole patch.

5 EXPERIMENTS & RESULTS

5.1 Comparisons

We begin with a comparison to various previous techniques for sweeping volume. Fig. 4 shows a comparison of our method to the ground truth exact solution, which can be computed in this case of a simple translation along a straight line. Our output is visually identical, and furthermore has a significantly smaller error compared to stamping (see Fig. 12). In this and all other examples, inputs are scaled to fit the unit cube.

In Fig. 6, we replicate the *pièce de résistance* from Zhang et al. [2009], for which our general method yields the same output. Using the same brush, we also show a sweep with a more elaborate trajectory. In Fig. 5, we qualitatively match [Rossignac et al. 2007]'s main result; they are limited to screw trajectories, while we show our method's output on other trajectories not possible with theirs.

In Fig. 3 we show a comparison to the *stamping* algorithm as well as to an "improved" stamping which, at each grid corner, performs a fixed number of gradient descent steps on Eq. (1) with equally spaced initial guesses. Fast moving or thin components in the input lead to geometric and topological artifacts, while our algorithm recovers those delicate parts perfectly. Furthermore, the number of stamps required to obtain a desirable output is not constant. As we show in Fig. 10, an adequate number of stamps at one grid resolution leads to staircasing artifacts in the other. Stamping more densely in time alleviates that issue at the unacceptable price of a high number of queries, significantly increasing the computational cost (see Fig. 11).

5.2 Geometric modeling via sweeping

Our method's robustness and generality allows exploration of artistic modeling using sweeps, as shown in Fig. 15. In Fig. 13 we model the horn of a gramophone using a sweep of a 2D curve.

Our method is extremely general and opens up the option for many applications– we can work on any time-evolving signed distance field (SDF), as long as it is differentiable with respect to time. Fig. 14 shows an analytic SDF of a torus evolved from the classic

Fig. 16. We capture trajectories with a real VR setup and then sweep different letters' mesh representations along them.

Fig. 17. We sweep a ballet dancer's motion, making use of scaling and transparency (the latter, mapped to $t^{\star}$) to resemble an artistic motion trail. 3D model by Maryam Sadeghi under CC BY-SA 3.0.

circular torus at the bottom of the sweep, to a square L_1-torus at the top. The result is a vase with a round bottom and a square top.

Similarly, any trajectory can be used as long as it is differentiable with respect to time: In Fig. 16, we interpolated between frames captured by a virtual reality sculpting application, using Camtull-Rom splines for translation and spherical linear interpolation for rotation. Composing the rigid motion with time-varying scaling of the brush is also easy to incorporate, as shown in Fig. 17.

5.3 CSG operations on sweeps

Beyond sweeps, our method can fit within other constructive solid geometry (CSG) operations with sweeps. For example, in Fig. 18 we subtract a sweep S generated by a moving brush $B(t)$ of the Artificially Flavored Drink-Mix Man from an implicitly represented solid mountain M. Naively, to get a new mountain with a hole punched through it, $N = M \setminus S$, we could run our method to compute S and then use a standard Boolean operation to perform the subtraction. However, most of the sweep is far from the mountain and, thus, does not affect the resulting N, so there is need to compute it all. Instead, we can turn the order of operations on its head — we define the time-dependent implicit function $R(t) = \min(-M, B(t))$. We run our continuation method with this implicit function directly. Fig. 18 shows the total volume actually computed by our in green, exhibiting its efficiency. The full red swept volume on the left is shown only for trajectory visualization purposes.

Fig. 18. Most of the character's swept volume does not intersect the mountain. Thanks to our continuation algorithm, we can directly contour the CSG subtraction without wasting computational time on the irrelevant parts of the swept volume. Model by Perry Engel under CC BY-NC 4.0.

Fig. 19. The intersection between the path of a spaceship and a rock formation is computed without necessary computing the swept volume of the entire path, cutting runtime by a factor of 20. Model by Karl, CC BY-SA 3.0.

In Fig. 19 we show the computed intersection points of a Space Wizard Vehicle's path as it is attempting a risky maneuver through a canyon. Only the red highlighted parts were used in the computation of the intersection, speeding up the wall-clock computational time by a factor of 15.6×: 344M distance queries (full swept volume) down to just 24M queries (just intersection). The swept volume on the left is shown only for trajectory visualization purposes.

In Fig. 20 the sweep of a pendulum is used to design a shape (pink cylinder) that can rotate while letting the pendulum pass perfectly pass through it. We perform a change of frame of reference so that the pink ring is kept fixed and the rest of the world is rotating around it (on top of the pendulum motion). We then compute the subtraction of the pendulum from the pink ring directly.

Of course, CSG operations are useful operations for e.g., modeling, such as carving a pumpkin using various brushes, as shown in Fig. 2.

5.4 Path planning

Sweeps can be useful for inferring that total volume that may be potentially occupied by a moving object. In Fig. 21, the joint transformations of each rigid component are composed to create a complex transformation. Our smooth swept volume reveals the space occupied by this robotic arm: that is, the areas one should stay clear of to avoid getting whacked.

In Fig. 23, we precompute the parking maneuvers of a car, to yield a sweep that could guide other vehicles that aim to leave a path for the car. In Fig. 24 we compute the Minkowski sum of a 3D printer's head with a rectangle, so as to infer the total volume its various motions may take up. To accomplish this, we generalize our minimization over 1D time t to the 2D rectangles uv coordinates. While existing methods for exactly computing Minkowski sums of triangle meshes often lead to expensive pruning and intersection resolving operations (cf. [Campen and Kobbelt 2010; Cherchi et al. 2020]), our algorithm contours the zero level set directly.

Fig. 20. Changing the frame of reference to that of a fixed wheel allows us to produce this result as a CSG subtraction of a single swept volume.

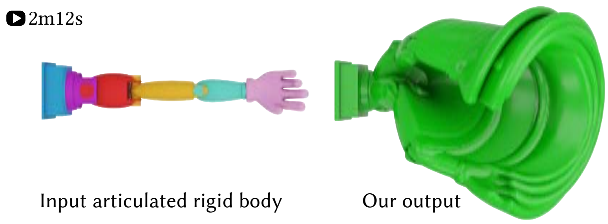


Fig. 21. An articulated rigid body can be seen as the union of several solids moving with different trajectories. As such, it fits perfectly into our CSG framework and we can compute its swept volume.

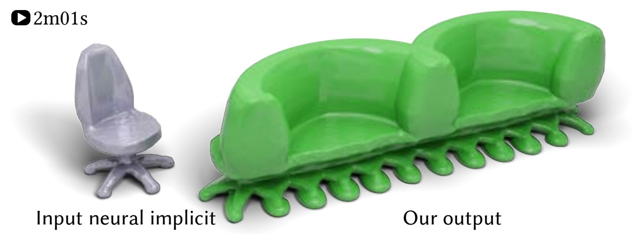


Fig. 22. Sweeping a neural network reconstruction of a signed distance field obtained via [Davies et al. 2021].

5.5 Sweeps on generalized implicit functions

Our method is readily applicable to other implicit functions. In Fig. 22, the input is a neural implicit function reconstruction of a chair [Davies et al. 2021]. Likewise, generalized winding number [Barill et al. 2018] enables generating an implicit function from a point cloud as shown in Fig. 25. As a matter of fact, some of our input triangle meshes (like the car in Fig. 23 and the space shuttle in Fig. 1) are non solid models made up of intersecting components, but our use of the winding number-signed distance field makes our algorithm robust to those intersections.

5.6 Timing and implementation details

We implemented our method in C++, relying heavily on the library LIBIGL [Jacobson et al. 2018]. We report timings conducted on a 2020 MacBook Pro with a 2.3 GHz Quad-Core Intel Core i7 processor and 16 GB of memory. The main bottleneck in our algorithm is the querying of $f(x, t)$ during the gradient descent, consistently taking up over 95% of our runtimes. We share this with the *stamping* algorithm, to which we compare performance-wise in Fig. 10 and Fig. 11. The reduction in asymptotic complexity makes our algorithm faster than stamping at a resolution which produces glaring aliasing artifacts (see Fig. 10, bottom left).

5.7 Surface extraction

In our results, we use Ju et al.'s dual contouring method for surface extraction [2002]. Similar to other surface extraction methods [Bloomenthal 1988; Kobbelt et al. 2001; Lorensen and Cline 1987], it considers each grid edge $(\mathbf{x}_1, \mathbf{x}_2)$ crossing the zero level set, with implicit values $f_1 < 0 < f_2$ and performs a local search to find the point $\mathbf{x}_s$ on which the edge crosses the zero level set ($f^\star(\mathbf{x}_s) = 0$). Dual contouring then uses position and gradient information at this point to compute a dual vertex lying in the edge's neighboring (primary) voxel cells. This combines seamlessly with our method: we keep the argmins $t_1^\star, t_2^\star$ and when the binary search asks for the implicit values $f^\star$ of point $\mathbf{x}_s$, we perform gradient descent over $f(\mathbf{x}_s, t)$, initializing from both $t_1^\star, t_2^\star$ and choosing the best result.

Dual contouring the spatial gradient $\partial f^\star / \partial \mathbf{x}$ (surface normal up to length) at the identified level-set points $\mathbf{x}_s$. In our case,

$$\frac{d\, f^\star}{d\, \mathbf{x}} = \frac{\partial f^\star}{\partial \mathbf{x}} + \frac{\partial t^\star}{\partial \mathbf{x}} \frac{\partial f^\star}{\partial t^\star} \tag{3}$$

Being at a minima $t_s^\star$, the rightmost term must vanish. Thus it is sufficient to compute only the spatial gradient of f evaluated at time $t_s^\star$. Occasionally, $\mathbf{x}_s$ may fall on the medial axis of our brush at $t_s^\star$, where f is non-differentiable and does not admit a gradient. In such cases, we find finite differencing will be more reliable for computing these gradients. We observe this rarely for strictly positive level sets, so our dual contouring uses a numerically tiny offset $\varepsilon > 0$.

6 CONCLUSION

We believe the robust and efficient method for computation of swept volumes introduced in this paper opens up possibilities for future work, such as handling sweeps of non-rigid deformations, or further applications in modeling and path-planning.

We are still unsatisfied with the speed of our method. Even though it surpasses other general techniques in its performance, it is still too slow to be used in interactive applications for complex inputs in which real time feedback is required. While stamping can work on a fine grid at real time at the cost of reducing *temporal* resolution (thereby creating aliasing), our method is continuous in time. One option to overcome this would be parallelization on the GPU, though less trivial than stamping's "embarrassingly parallel" nature.

Our continuation assumes that the input brush is a solid with a well-defined interior and exterior, hence we cannot apply our method directly to infinitely thin sheets, curves or non-orientable surfaces without introducing a finite offset.

In our experiments, our algorithm never failed to recover the correct swept volume surface, even when facing pathological trajectories (e.g., Fig. 8), pathologically complex inputs with high genus (e.g., Fig. 26), very low or high grid size (e.g., Fig. 10) or very low number of initial queue seeds (e.g., Fig. 9). On the theoretical side, however, we note that we do not have a formal proof of correctness. We set it as important future work related to formal guarantees on

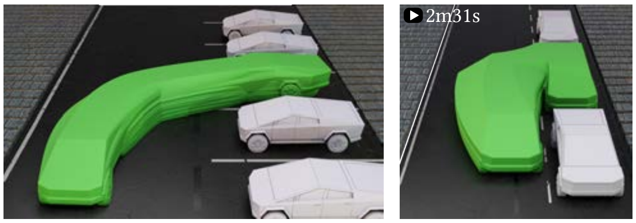


Fig. 23. The swept volume by a car performing parking maneuvers could aid autonomous decision making. Model by W. Mckay, CC BY-NC 4.0.

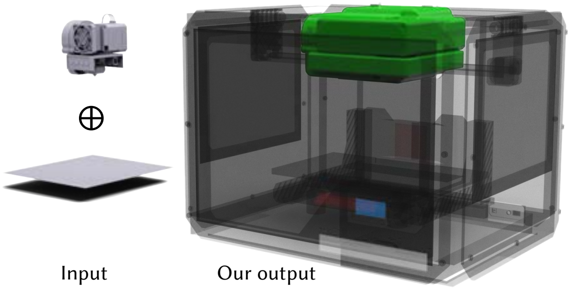


Fig. 24. We compute the region of space covered by a 3D printer's head by a Minkowski sum with a rectangle to aid design of the printer's inside.

global root finding methods. A deeper theoretical study of the one-dimensional $f(x,t)$ functions and, more critically, a characterization of the discontinuities in the $t^\star$ function would not only help in formalizing our method's robustness, but ideally also in optimizing the choice and number of seeds in our continuation.

Lastly, we believe that many applications concerning time-evolving implicits and their level sets, such as fluid simulation, can benefit from our core observation, of considering the argmin-manifold continuation - we have just scratched the surface of what is possible with this general technique. To be continued!

ACKNOWLEDGMENTS

This project is funded in part by NSERC Discovery (RGPIN2017–05235, RGPAS–2017–507938), New Frontiers of Research Fund (NFRFE–201), the Ontario Early Research Award program, the Canada Research Chairs Program, the Fields Centre for Quantitative Analysis and Modelling and gifts by Adobe Systems, Autodesk and MESH Inc.

We thank Oded Stein, Abhishek Madan and Rahul Arora for their help rendering our results; Thomas Davies and David Farrell for providing us the inputs for Fig. 22 and Fig. 16, respectively; Xuan Dam, John Hancock and all the University of Toronto Department of Computer Science research, administrative and maintenance staff that literally kept our lab running during a very hard year.

REFERENCES

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.

Steven Abrams and Peter K. Allen. 2000. Computing swept volumes. *Comput. Animat. Virtual Worlds* (2000).

Eugene L Allgower and Kurt Georg. 2003. *Introduction to numerical continuation methods.* SIAM.

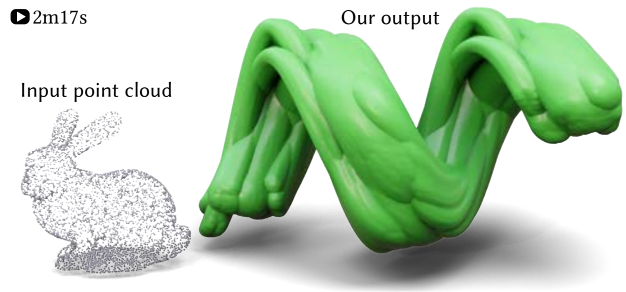


Fig. 25. Our use of the generalized winding number from [Barill et al. 2018] to provide a sign to a distance field means we can even compute the "swept volume" of a point cloud's isolevel.

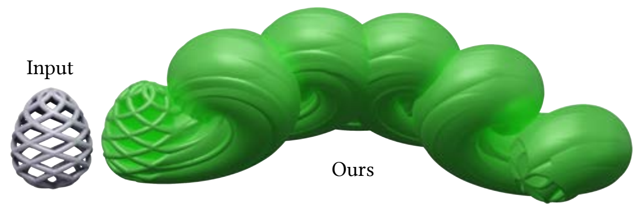


Fig. 26. A stress test of our algorithm containing many intersections. 3D model by Mariano Mumpower under CC BY-SA 3.0.

Gavin Barill, Neil Dickson, Ryan Schmidt, David I.W. Levin, and Alec Jacobson. 2018. Fast Winding Numbers for Soups and Clouds. *ACM Transactions on Graphics* (2018).

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

Jules Bloomenthal. 1988. Polygonization of implicit surfaces. *Computer-Aided Geometric Design* (1988).

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization.*

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

Gianmarco Cherchi, Marco Livesu, Riccardo Scateni, and Marco Attene. 2020. Fast and Robust Mesh Arrangements Using Floating-Point Arithmetic. *ACM Trans. Graph.* (2020).

Thomas Davies, Derek Nowrouzezahrai, and Alec Jacobson. 2021. On the Effectiveness of Weight-Encoded Neural Implicit 3D Shapes. arXiv:2009.09808 [cs.GR]

Akash Garg, Alec Jacobson, and Eitan Grinspun. 2016. Computational Design of Reconfigurables. *ACM Trans. Graph.* 35, 4 (2016).

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust Inside-Outside Segmentation using Generalized Winding Numbers. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)* 32, 4 (2013), 33:1–33:12.

Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe D. Warren. 2002. Dual contouring of hermite data. *ACM Trans. Graph.* 21, 3 (2002), 339–346.

Young J Kim, Gokul Varadhan, Ming C Lin, and Dinesh Manocha. 2004. Fast swept volume approximation of complex polyhedral models. *Computer-Aided Design* 36, 11 (2004), 1013–1027.

Leif P Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. 2001. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques.* 57–66.

William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics* 21, 4 (1987).

Jai Menon, Richard J Marisa, and Jovan Zagajac. 1994. More powerful solid modeling through ray representations. *IEEE Computer Graphics and Applications* 14, 3 (1994).

Jorge Nocedal and Stephen Wright. 2006. *Numerical optimization.* Springer.

Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proc. CVPR.*

Darko Pavic and Leif Kobbelt. 2008. High-Resolution Volumetric Computation of Offset Surfaces with Feature Preservation. *Comput. Graph. Forum* (2008).

Martin Peternell, Helmut Pottmann, Tibor Steiner, and Hongkai Zhao. 2005. Swept volumes. *Computer-Aided Design and Applications* 2, 5 (2005), 599–608.

Íñigo Quílez. 2020. Interior SDFs. https://www.iquilezles.org/www/articles/interiordistance/interiordistance.htm. Accessed: 2021-05-12.

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* (2007).
Ryan M. Schmidt and Brian Wyvill. 2005. Generalized sweep templates for implicit modeling. In *Proc. GRAPHITE*. 187–196.
William J Schroeder, William E Lorensen, and Steve Linthicum. 1994. Implicit modeling of swept surfaces and volumes. In *Proceedings Visualization'94*. IEEE, 40–45.
A. Sourin and A. Pasko. 1995. Function Representation for Sweeping by a Moving Solid. In *Proc. Symp. on Solid Modeling and Applications*.
Ugur Sungurtekin and H Voelcker. 1986. Graphical simulation & automatic verification of NC machining programs. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, Vol. 3. IEEE, 156–165.
Andreas Von Dziegielewski, Michael Hemmer, and Elmar Schömer. 2012. High quality conservative surface mesh generation for swept volumes. In *2012 IEEE International Conference on Robotics and Automation*. IEEE, 764–769.
W. P. Wang and K. K. Wang. 1986. Geometric Modeling for Swept Volume of Moving Solids. *IEEE Computer Graphics and Applications* 6, 12 (1986).
John D. Weld and Ming C. Leu. 1990. Geometric Representation of Swept Volumes with Application to Polyhedral Objects. *Int. J. Robotics Res.* (1990).
Geoff Wyvill, Craig McPheeters, and Brian Wyvill. 1986. Soft objects. In *Advanced Computer Graphics*. Springer, 113–128.
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 Trans. Graph.* (2016).

A THEORETICAL CONVERGENCE

In what follows, assume a shape $\mathcal{S}$ is being swept along a trajectory $\mathcal{S}(t)$ with $0 \leq t \leq 1$ whose velocity and acceleration are bounded in norm from above by v_b, a_b. Let $\text{sd}(P, \mathcal{S}(t))$ be the SDF of $\mathcal{S}(t)$ measured at point P, and further assume the SDF is twice-differentiable with respect to t for all P. Let

$$\text{sd}_{gt}(P) = \min_{0 \leq t \leq 1} \text{sd}(P, \mathcal{S}(t)) \tag{4}$$

be the groundtruth swept volume SDF,

$$\text{sd}_{stamp,n}(P) = \min_{i \in \{0,\dots,n\}} \text{sd}(P, \mathcal{S}(i/n)) \tag{5}$$

be the stamping algorithm SDF with n stamps and

$$\text{sd}_{ours,n,m}(P) = \min_{i \in \{0,\dots,n\}} g_m(\text{sd}(P, \mathcal{S}(i/n))) \tag{6}$$

be a conservative version of our algorithm which samples uniformly in time and then carries out m iterations of backtracking gradient descen (g_m) t for each of the samples. Also, let τ be the length of the minimum interval in which $\text{sd}(P, \mathcal{S}(t))$ is strongly convex and Lipschitz-smooth as a function of t.

Lemma A.1. *In the above conditions,*

$$\left\|\text{sd}_{gt} - \text{sd}_{stamp,n}\right\|_\infty \leq \frac{v_b}{2n} \tag{7}$$

Proof. Let P be a point in space and $t^* = \operatorname{argmin} \text{sd}(P, \mathcal{S}(t))$. Let i be the closest uniform timestep such that $|i/n - t^*| \leq 1/2n$. Then,

$$|\text{sd}(P, \mathcal{S}(i/n)) - \text{sd}(P, \mathcal{S}(t^*))| \leq v_b \frac{1}{2n}\,. \tag{8}$$

□

Lemma A.2. *In the above conditions, if $n > 1/\tau$, there exists a constant $K \in \mathbb{R}$ and a $c > 1$ such that for big enough m,*

$$\left\|\text{sd}_{gt} - \text{sd}_{ours,n,m}\right\|_\infty \leq K \frac{1}{\tau^2 c^m} \tag{9}$$

Proof. Let P be a point in space and $t^* = \operatorname{argmin} \text{sd}(P, \mathcal{S}(t))$. Let i/n one of the uniform sample which falls on the interval on which the function is convex and which contains t^* (there is at least one such i because $n > 1/\tau$). Now, from the convergence of a gradient descent with backtracking linesearch (see [Nocedal and Wright 2006]) under sufficient regularity conditions,

$$|\text{sd}(P, \mathcal{S}(t^*)) - \text{sd}(P, \mathcal{S}(t))| \leq K \frac{|t^* - i/n|^2}{c^m} \leq K \frac{1}{c^m \tau^2} \tag{10}$$

□

Theorem A.3. *Under the regularity conditions described above, guaranteeing*

$$\left\|\text{sd}_g\, t - \text{sd}_{stamp,n}\right\| \leq \varepsilon \tag{11}$$

requires $O(1/\varepsilon)$ evaluations of $\text{sd}(P, \mathcal{S}(t))$, while guaranteeing

$$\left\|\text{sd}_g\, t - \text{sd}_{ours,n,m}\right\| \leq \varepsilon \tag{12}$$

requires $O(\log(1/\varepsilon))$ evaluations of $\text{sd}(P, \mathcal{S}(t))$.

Proof. The first statement is a direct consequence of Lemma A.1, while the second comes from Lemma A.2, making $n = \lceil 1/\tau \rceil$ and $m = \log(K/\varepsilon\tau^2)$, leading to

$$\frac{1}{\tau} \log_c \frac{K}{\varepsilon\tau^2} \tag{13}$$

function evaluations. □

B PSEUDOCODE

Algorithm 1: argmin Continuation Method

let f_i, t_i be the stored implicit and time values at corner i
Insert voxel and time seeds into Q
while *Q is not empty* **do**
  $v, t_v \leftarrow$ pop voxel and time from Q
  for *each corner i of the eight corners of v* **do**
    $f_i, t_i \leftarrow$ backtracking gradient descent from t_v at $\mathbf{x}_i$
    if *first time seeing i* **then**
      $f_i^\star, t_i^\star \leftarrow f_i, t_i$
    else if $f_i < f_i^\star$ **then**
      $f_i^\star, t_i^\star \leftarrow f_i, t_i$
      for *each other voxel n incident on i* **do**
        push (n, t_i) onto Q
    else
      push $(v, t_i^\star)$ onto Q
  if *any corner was updated* **then**
    for *each edge $\{i, j\}$ of the twelve corners of v* **do**
      if *signs of $f_i^\star$ and $f_j^\star$ differ* **then**
        for *each other voxel n incident on $\{i, j\}$* **do**
          push $(n, t_i^\star)$ onto Q
          push $(n, t_j^\star)$ onto Q