Fast Swept Volume Approximation of Complex Polyhedral Models

http://gamma.cs.unc.edu/SV

Young J. Kim Gokul Varadhan Ming C. Lin Dinesh Manocha
Department of Computer Science, UNC-Chapel Hill
{youngkim,varadhan,lin,dm}@cs.unc.edu

ABSTRACT

We present an efficient algorithm to approximate the swept volume (SV) of a complex polyhedron along a given trajectory. Given the boundary description of the polyhedron and a path specified as a parametric curve, our algorithm enumerates a superset of the boundary surfaces of SV. It consists of ruled and developable surface primitives, and the SV corresponds to the *outer boundary* of their arrangement. We approximate this boundary by using a five-stage pipeline. This includes computing a bounded-error approximation of each surface primitive, computing unsigned distance fields on a uniform grid, classifying all grid points using fast marching front propagation, iso-surface reconstruction, and topological refinement. We also present a novel and fast algorithm for computing the signed distance of surface primitives as well as a number of techniques based on surface culling, fast marching level-set methods and rasterization hardware to improve the performance of the overall algorithm. We analyze different sources of error in our approximation algorithm and highlight its performance on complex models composed of thousands of polygons. In practice, it is able to compute a bounded-error approximation in tens of seconds for models composed of thousands of polygons sweeping along a complex trajectory.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling

General Terms

Algorithms, Design

Keywords

Swept Volume, Implicit Modeling, Distance Fields

1. INTRODUCTION

Swept volume (SV) is the volume generated by sweeping a solid or a collection of surfaces in space along a smooth trajectory. The problem of SV computation arises in different applications, including NC machining verification [11, 13], geometric modeling [16, 37], robot workspace analysis [4, 7], collision detection [30, 51], maintainability study [34], ergonomic design [3], motion planning [42], etc. A more extensive list of potential applications of SV can be found at [1].

The SV computation problem has been studied in different disciplines for more than four decades. This includes elegant work based on envelope theory, singularity theory, Lie groups, sweep differential equations on the characterization of the problem. As a result, the mathematical formulation of SV computation is relatively well-understood.

In many applications, the main goal of SV computation is to identify and extract the boundary of the SV, in particularly its outermost boundary. Most of the algorithms for computation of the boundary of SV are based, either explicitly or implicitly, on the following framework:

1. Find all the boundary primitives that contribute to the outermost boundary of SV.
2. Compute an arrangement of the boundary primitives by performing intersection and trimming computations.
3. Traverse the arrangement and extract the outer boundary. Here, the outer boundary of an arrangement is defined as the boundary of a cell, which is reachable from infinity following some continuous path, in the arrangement.

Most of the mathematical work has mainly dealt with characterizing the boundary primitives, given some assumptions on the sweeping path. There is a considerable amount of research in computational geometry on the combinatorial complexity of computing arrangements as well as on surface-surface intersection computations in geometric and solid modeling. However, the underlying combinatorial and algebraic complexity of exact SV computation is very high. Furthermore, the implementations of any algorithms for computing intersections and arrangements need to deal with accuracy and robustness issues. As a result, no practical algorithms are known for exact computation of the SV for any arbitrary polyhedron sweeping along a given smooth path.

Given the underlying complexity of exact SV computation, most of the earlier work has focussed on approximate techniques. Different algorithms can be characterized based on whether they are limited to 2D objects, or they only compute an image-space projection or visualization of the SV from a given viewpoint, or compute a relatively coarse

SM'03, June 16–20, 2003, Seattle, Washington, USA.

discretization of the boundary primitives followed by union computation of different configurations of the polyhedra along the trajectory. These algorithms are either slow for practical applications, or suffer from robustness problems, or compute a rather coarse approximation of the SV.

Main Results

We present an efficient algorithm to approximate the outermost boundary of SV's of complex polyhedral models along a given trajectory. The algorithm initially enumerates a superset of the boundary primitives of SV, which consists of *ruled* and *developable* surfaces [48]. The ruled surface is generated by considering each edge in the original model as a ruling line and the trajectory as a directrix curve. The developable surface is obtained by applying the envelope theory to moving triangles. Given a formulation of the boundary elements, our algorithm computes an approximation to the resulting arrangement using a five-stage pipeline. Firstly, it computes a bounded-error polygonal approximation of each surface primitive. Secondly, it samples the surface primitives by computing unsigned, directed distance fields along the vertices of a grid. Next it classifies the grid points to be either inside or outside of the surfaces to obtain the signed distance field using a novel algorithm based on marching front propagation. This is followed by iso-surface reconstruction. Finally, the algorithm performs topological refinement, taking into account some of the characterizations of the SV computation. We also present a number of acceleration techniques based on culling of surface primitives, use of interpolation-based rasterization hardware for fast computation of distance field, and a variation of fast marching level-set method for classification of grid points.

Our algorithm computes a bounded-error approximation of the SV and we analyze all sources of error. We have implemented this algorithm on a commodity-based PC with nVidia GeForce 4 graphics card, and benchmarked its performance on complex benchmarks. The underlying polyhedral models consist of thousands of triangles and are sweeping along a complex trajectory corresponding to a parametric curve. The computation of SV takes a few tens of seconds on a 2.4GHz Pentium IV processor.

As compared to earlier approaches, the main advantages of our technique include:

- Generality: The algorithm can handle general 2-manifold polyhedral models, and makes no assumptions about the sweep path.
- Complex Models: The algorithm is directly applicable to complex models composed of a high number of features. Given a trajectory and a bound on the approximation error, the overall complexity increases as a linear function of the input size.
- Efficiency: The use of culling techniques and algorithms for signed distance field computation significantly improve the running time of the algorithm.
- Simplicity: The algorithm is relatively simple to implement and does not suffer from robustness problems or degeneracies.
- Good SV Approximation: Our preliminary application of the algorithm to different benchmarks indicates that it can compute a good, bounded-error approximation of the boundary.

Organization

The rest of our paper is organized as follows. In Section 2, we briefly review the earlier work on SV computation. Section 3 provides the overview of our approach to SV computation. In Section 4, we present an algorithm to compute the boundary surface primitives of SV. Section 5 describes our approximation algorithm to compute the arrangement of the surface primitives using sampling and reconstruction. We analyze the performance of our algorithm in Section 6 and describe its implementation and performance in Section 7. In Section 8, we conclude our paper and present future work.

2. PREVIOUS WORK

In this section, we give a brief survey of the work related to SV computation, arrangements, and iso-surface reconstruction based on distance fields.

2.1 Swept Volume Computation

SV has been studied quite extensively over the years. We list some of the crucial development in the history of SV research here, but refer the readers to see [1] for more thorough survey of SV-related work.

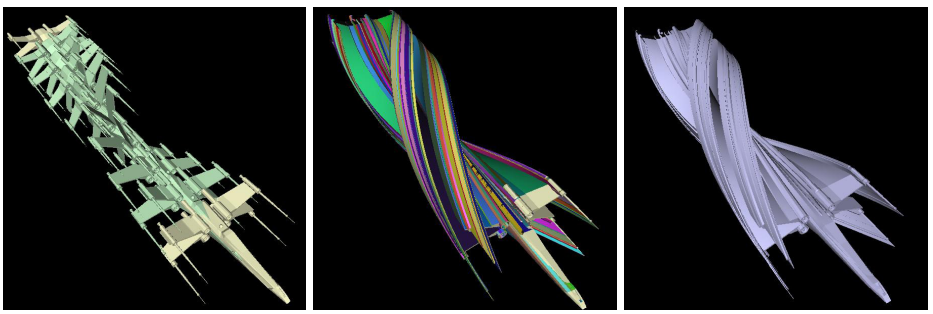
Methodology: The mathematical formulation of the SV problem has been investigated using singularity theory (or Jacobian rank deficiency method) [2, 5, 6], Sweep Differential Equation (SDE) [11, 12], Minkowski sums [19], envelope theory [38, 48], implicit modeling [41], and kinematics [29]. Moreover, most of this work deals with the SV of generic, free form objects.

Polyhedral Approximation: Given the complexity of computing the exact SV, few algorithms have been developed to provide a polyhedral approximation of SV. In 2D, [9, 35] study an approximation of the general sweep for curved objects, and they have been applied to font design. In 3D, [48] describe a geometric representation of SV for compact n-manifolds with application to polyhedral objects. [41] use discretized representations and iso-surface reconstruction to approximate SV, [7, 40] compute the arrangement of swept polyhedral surfaces based on their coarse approximation, [10] study a simple rotational sweep of exact SV. However, these 3D algorithms are either restricted to simple geometric primitives [40] or simple sweep trajectory [10], or suffer from accuracy [41] and robustness problems [7].

Visualization: Many algorithms have been proposed to visualize the boundary of the SV using the rasterization hardware. These algorithms use the Z-buffer hardware to compute a 2D projection of the surface from a given viewpoint and not the actual boundary of the 3D SV. [26, 27, 44, 47] utilizes rasterization hardware to simulate NC machining display, [16] uses the Jacobian rank deficiency method to visualize a SV of trivariate tensor-product B-spline solids, and [49] studies the SV computation of a 2D image.

2.2 Arrangement Computation

Given a finite collection of geometric objects in R^d, their arrangement is the decomposition of R^d into connected open cells [23]. The arrangement computation problem is ubiquitous by nature and arises in a number of applications. A survey of different algorithms and complexity bounds for ar-



(a) Trajectory (b) Surfaces (c) $\partial SV(\mathbf{\Gamma})$

Figure 1: Complexity of SV Computation. ***(a) shows a sweeping trajectory of a cubic polynomial curve for a X-Wing model. In (b), each surface primitive comprising in*** $\partial SV(\mathbf{\Gamma})$ ***(total 3793 surface primitives) is color-coded differently. (c) shows*** $\partial SV(\mathbf{\Gamma})$***, an outer boundary of the surface elements.***

rangements computations is given in [23].

Complexity: It is well known that the worst case combinatorial complexity of an arrangement of n surfaces in R^d is $O(n^d)$ [23], and there are such arrangements having $\theta(n^d)$ complexity, thus this bound is tight. In this analysis, each surface is assumed to have a bounded algebraic degree, and needs to be decomposed into *monotonic* patches as well.

Algorithms: There are quite a few known algorithms to compute an arrangement using both deterministic algorithms and randomized algorithms. This includes an output-sensitive algorithm to compute an arrangement of surfaces in 3-space and has $O(n\lambda_q(n)\log(n) + V\log(n))$ time complexity, where V is the combinatorial complexity of the vertical decomposition, q is a constant depending on the degree of the surfaces, and $\lambda_q(n)$ is the maximum length of (n, q) Davenport-Schinzel sequences [17].

Implementation Issues: Some of the major issues in the implementation of arrangement computation algorithms are accuracy and robustness problems. It is quite hard to enumerate all degenerate configurations, especially when the primitives are non-linear surfaces. [40] enumerate 15 different possible degenerate cases for an arrangement of polyhedral surfaces. Moreover, [40] proposed a controlled perturbation scheme, and applied it to polyhedral SV approximation. However, it can take a considerable amount of time for models composed of few hundred triangles. These problems get more severe when we are dealing with curved primitives.

2.3 Distance Field Computation and Iso-Surface Reconstruction

Recently, distance fields have been increasingly used in volumetric shape representation [21, 22], proximity computations based on rasterization hardware [25], path planning [31], surface metamorphosis [15], and SV computation [41].

Grid-based iso-surface reconstruction has been extensively studied beginning from the seminal work of the Marching Cubes algorithm [36], and has been extended to its variants such as the Enhanced Marching Cubes (EMC) [32] or the dual contouring method [28]. [50] have used surface wavefront propagation techniques to extract semi-regular meshes from volumes.

3. OVERVIEW

In this section, we characterize the mathematical formulation of computing the SV of general polyhedral models and also give an overview of our approximation scheme.

3.1 Notation

We use bold-faced letters to distinguish a vector (e.g. $\boldsymbol{p}(t)$) from a scalar value (e.g. time t). f, v, e respectively denotes a face, a vertex, and an edge of a polyhedron. We use $f_k^{\mathbf{\Gamma}}$ to denote the kth face of a polyhedron $\mathbf{\Gamma}$.

3.2 Problem Formulation

Let $\mathbf{\Gamma}$, also known as a generator, be a polyhedron in R^3. Let the sweep trajectory $\boldsymbol{\tau}(t)$ be a tuple of $(\mathbf{\Psi}(t),\ \boldsymbol{R}(t))$, where $\mathbf{\Psi}(t)$ is a time-varying, differentiable vector in R^3 and $\boldsymbol{R}(t)$ is a time-varying, orthonormal matrix in $SO(3)$. Here, both $\mathbf{\Psi}(t)$ and $\boldsymbol{R}(t)$ depend on a single variable, the time $t \in [0, 1]$. Furthermore, $\mathbf{\Psi}(0)$ corresponds to the origin, and $\boldsymbol{R}(0)$ to the identity matrix. Then, consider the following sweep equation of $\mathbf{\Gamma}(t)$:

$$\mathbf{\Gamma}(t) = \mathbf{\Psi}(t) + \boldsymbol{R}(t)\mathbf{\Gamma} \tag{1}$$

In our paper, the SV of the generator $\mathbf{\Gamma}$ along the trajectory $\boldsymbol{\tau(t)}$ is defined as:

$$SV(\mathbf{\Gamma}) = \{ \cup\, \mathbf{\Gamma}(t) \mid t \in [0, 1] \} \tag{2}$$

Notice that our SV equation is allowed with only rigid motions (i.e., translation and rotation), even though, in general, $\boldsymbol{\tau}(t)$ can be any isotopy mapping [48].

Our goal is to compute the boundary of $SV(\mathbf{\Gamma})$, $\partial SV(\mathbf{\Gamma})$, without internal voids. More formally, consider an arrangement $\mathcal{A}$ and a cell $\mathcal{C}$ in $\mathcal{A}$, which is reachable from infinity following some continuous path. Let us further define the *outer boundary* of $\mathcal{A}$ as the boundary of $\mathcal{C}$. Then, we want to compute the outer boundary[1] of $\mathcal{A}$ induced by the surface elements in $SV(\mathbf{\Gamma})$. We use the following theorems [48] to characterize the boundary of SV:

[1]Throughout the paper, we interchangeably use the outer boundary of $\mathcal{A}$ and the outer boundary of $SV(\mathbf{\Gamma})$ to describe $\partial SV(\mathbf{\Gamma})$.

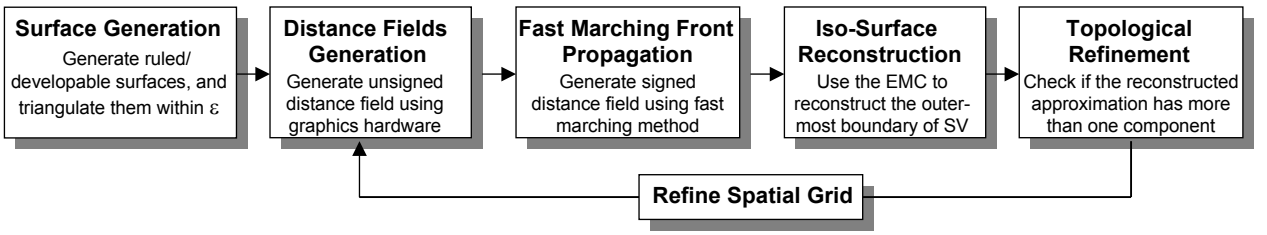


Figure 2: Our Swept Volume Computation Pipeline

THEOREM 3.1 *If during the sweep* $\mathbf{\Gamma(t_i) \cap \Gamma(t_j)} = \phi$ *for* $t_i \neq t_j$, *then* $SV(\mathbf{\Gamma}) = \{ \cup_{k=1}^{n} SV(f_k^{\mathbf{\Gamma}}) \mid n$ *is the number of faces in* $\Gamma \}$.

THEOREM 3.2 $SV(f_k^{\mathbf{\Gamma}})$ *consists of:*

- *Faces in* $f_k^{\mathbf{\Gamma(0)}}$ *and* $f_k^{\mathbf{\Gamma(1)}}$
- *Ruled surfaces using the edges of* $f_k^{\mathbf{\Gamma}(t)}$ *as a ruling line along the directrix* $\boldsymbol{\tau}$
- *Developable surfaces as an envelope of* $f_k^{\mathbf{\Gamma}(t)}$ *along* $\boldsymbol{\tau}$.

Therefore, computing the boundary of $\partial SV(\mathbf{\Gamma})$ boils down to computing ruled and developable surface primitives, and finally computing the outer boundary of their arrangement.

3.3 Approximation Algorithm

Our goal is to compute the outermost boundary, $\partial SV(\mathbf{\Gamma})$ where the complexity of $\mathbf{\Gamma}$ is relatively high, e.g. thousands of triangles. The major difficulty of the computation lies in the arrangement computation, as its computational and combinatorial complexity can be super-quadratic and its implementation is rather non-trivial due to the accuracy and robustness problems. Given the complexity of surface-surface intersection problem, it is very hard to robustly compute all the intersections between thousands of ruled and developable surface primitives within a reasonable time. For example, in Fig. 1, in order to exactly compute the SV of the X-Wing model consisting of 2496 triangles, we need to compute an arrangement of 3793 surfaces including calculating their intersection curves of as high as degree nine. Thus, instead of computing $\partial SV(\mathbf{\Gamma})$ exactly, we approximate it using an implicit modeling technique based on discretized representations and iso-surface-based reconstruction methods.

The main idea of our approximation approach is to compute the polyhedral approximation of ruled and developable surface primitives, generate their signed distance field, and reconstruct the outer boundary of the arrangement of the discretized surfaces. To accelerate this pipeline, we prune redundant surfaces in $SV(f_k^{\mathbf{\Gamma}})$, and perform fast distance field computation. As a result, the basic steps of our algorithm are as follows:

1. Given an error threshold of Hausdorff distance ϵ, we formulate the ruled and developable surfaces for each $SV(f_k^{\mathbf{\Gamma}})$, and compute a triangular approximation that is within the surface deviation error threshold. A subset of the primitives $SV(f_k^{\mathbf{\Gamma}})$ that do not contribute to the final boundary, $\partial SV(\mathbf{\Gamma})$ can be pruned away using sufficient criteria described in Sec. 4.3.
2. We compute the directed unsigned distance fields for each $SV(f_k^{\mathbf{\Gamma}})$ on a uniform 3D grid, using interpolation-based rasterization hardware.
3. We use a variant of the fast marching level set method to classify all the grid points whether they are inside or outside with respect to $\partial SV(\mathbf{\Gamma})$. This gives us a signed distance field.
4. Perform the iso-surface extraction on the resulting signed distance field to reconstruct the outermost boundary, $\partial SV(\mathbf{\Gamma})$
5. Perform a topological check to see if the reconstructed approximation has more than one component. If yes, we refine the spatial grid and perform the steps 2-5 again.

The above pipeline is illustrated in Fig. 2.

4. SURFACE GENERATION

In this section, we present techniques to compute the candidate surface primitives that contribute to the boundary of SV and compute a bounded error triangulation of each primitive. We also present new techniques to cull away surface primitives that do not compute the outer boundary of SV.

4.1 Boundary Surfaces

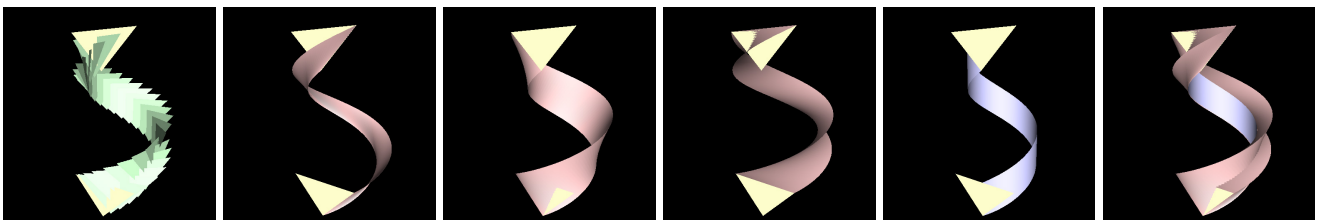
As shown in Thm. 3.1 in Sec. 3.2, the boundary of SV is obtained by computing the SV's of individual faces, $SV(f_k^{\mathbf{\Gamma}})$, in $\mathbf{\Gamma}$, and computing their union. Moreover, Thm. 3.2 states that, besides the trivial surfaces of $f_k^{\mathbf{\Gamma}}$ at initial and final positions during sweep (i.e., $\mathbf{\Gamma(0)}$ and $\mathbf{\Gamma(1)}$ in Eq. 1), there are only two types of surfaces that belong to $f_k^{\mathbf{\Gamma}}$: ruled and developable surfaces (also see Fig. 3).

4.1.1 Ruled Surface Primitives

A ruled surface is generated by sweeping a ruling line along a directrix curve. The surface $\boldsymbol{x}(u, v)$ has the following form:

$$\boldsymbol{x}(u, v) = \boldsymbol{b}(u) + v\boldsymbol{\delta}(u) \tag{3}$$

Here, $\boldsymbol{b}(u)$ is a directrix and $\boldsymbol{\delta}(u)$ is the direction of a ruling line. When we sweep $f_k^{\mathbf{\Gamma}}$ along the trajectory $\boldsymbol{\tau}(t)$, each edge e in $f_k^{\mathbf{\Gamma}}$ generates a ruled surface $\boldsymbol{x}(u, v)$. We denote the endpoints of an edge $\boldsymbol{e}$ by $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$. By substituting $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ for $\mathbf{\Gamma}$ in Eq. 1, we generate two curves, $\boldsymbol{b_0}(u)$ and $\boldsymbol{b_1}(u)$. Then, in Eq. 3, $\boldsymbol{b}(u)$ becomes $\boldsymbol{b_0}(u)$, and $\boldsymbol{\delta}(u)$ becomes $\boldsymbol{b_1}(u) - \boldsymbol{b_0}(u)$.



(a) Trajectory (b) Ruled 1 (c) Ruled 2 (d) Ruled 3 (e) Developable (f) SV

Figure 3: Boundary Surfaces of the SV of a Triangle. ***(a) shows a trajectory for the helical sweep of a yellow triangle. (b), (c), and (d) show ruled surfaces generated by the sweep, and (e) shows a developable surfaces by the sweep. (f) shows the final boundary surface of the sweep.***

4.1.2 Developable Surface Primitives

When a plane moves continuously along a trajectory $\boldsymbol{\tau}(t)$, its envelope generates a developable surface. Intuitively, a developable surface is a surface which can be made of a piece of paper [39]. Thus, a developable surface is locally isometric to a plane, and its Gaussian curvature at regular points is zero. Furthermore, a developable surface is a subset of a ruled surface.

Let us parametrically represent a moving plane $\boldsymbol{p}(u,v,t)$ as:

$$\boldsymbol{p}(u,v,t) = \boldsymbol{q}(t) + u\boldsymbol{r_1}(t) + v\boldsymbol{r_2}(t) \tag{4}$$

where $\boldsymbol{q}(t)$ is the origin of $\boldsymbol{p}(u,v,t)$, and $\boldsymbol{r_1}(t)$ and $\boldsymbol{r_2}(t)$ are two linearly independent vectors spanning $\boldsymbol{p}(u,v,t)$. Then, using the envelope theory, by solving $det(J(\boldsymbol{p}(u,v,t))) = 0$ for u and substituting the result for $\boldsymbol{p}(u,v,t) = 0$, we get the developable surface $\boldsymbol{d}(t,v)$ as [48]:

$$\begin{aligned} \boldsymbol{d}(t,v) &= \boldsymbol{b}(t) + v\boldsymbol{\delta}(t), \quad \text{where} \\ \boldsymbol{b}(t) &= \boldsymbol{q}(t) - \boldsymbol{r_1}(t)\frac{\boldsymbol{q'}(t)\cdot\boldsymbol{r_1}(t)\times\boldsymbol{r_2}(t)}{\boldsymbol{r_1'}(t)\cdot\boldsymbol{r_1}(t)\times\boldsymbol{r_2}(t)} \\ \boldsymbol{\delta}(t) &= \boldsymbol{r_2}(t) - \boldsymbol{r_1}(t)\frac{\boldsymbol{r_2'}(t)\cdot\boldsymbol{r_1}(t)\times\boldsymbol{r_2}(t)}{\boldsymbol{r_1'}(t)\cdot\boldsymbol{r_1}(t)\times\boldsymbol{r_2}(t)} \end{aligned} \tag{5}$$

This derivation is valid only if $\boldsymbol{r_1'}(t)\cdot\boldsymbol{r_1}(t)\times\boldsymbol{r_2}(t) \neq 0$. Otherwise, we can derive a similar equation in terms of u and t by getting rid of v in Eq. 4.

In the SV computation, sweeping $f_k^{\boldsymbol{\Gamma}}$ also generates a developable surface. Let us assume that $\boldsymbol{\Gamma}$ is triangulated, and denote any two edges of $f_k^{\boldsymbol{\Gamma}}$ by $\boldsymbol{e_1}$ and $\boldsymbol{e_2}$. Then, the direction vectors of $\boldsymbol{e_1}$ and $\boldsymbol{e_2}$ become $\boldsymbol{r_1}(t)$ and $\boldsymbol{r_2}(t)$ in Eq. 5. However, since Eq. 5 is derived from a plane, not from a triangle, the developable surface obtained from Eq. 5 needs to be clipped against the parametric domain of $\{u = 0, 0 \leq v \leq 1\}$, $\{0 \leq u \leq 1, v = 0\}$, and $\{u \geq 0, v \geq 0, u + v = 1\}$ for all t.

4.2 Bounded Error Triangulation

Once we have generated parametric representations for ruled and developable surface primitives, the next step is to compute a triangular approximation within a user-provided error deviation ϵ. There are many known algorithms for triangulating a rational parametric surface using either uniform [33] or adaptive tessellation [14, 46]. Since developable surfaces have zero Gaussian curvature, the uniform tessellation serves the purpose well; however, depending on the sweep trajectory $\boldsymbol{\tau}(t)$, the ruled surface can have regions of high curvature. In this case, the uniform tessellation tend to oversample the surface, so that the adaptive tessellation is more suitable. Notice that, depending on the chosen type of the trajectory $\boldsymbol{\tau}$, the ruled and developable surfaces can be well-known rational parametric surfaces or general parametric surfaces including trigonometric terms. However, since we can always perform a *flat-ness* test for smooth surface patches on the ruled and developable surfaces, we use a simple recursive algorithm like [14] to handle the general parametric surfaces as long as they are smooth surfaces.

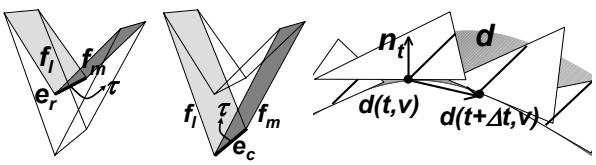
On the other hand, taking advantage of the nature of line geometry in ruled surfaces, one can also devise a *variational interpolatory subdivision* scheme for ruled surfaces [39]. Here, one recursively subdivides a ruled surface by minimizing an discrete energy functional that is represented in terms of an discrete approximation of mean curvatures at points on the surface.

4.3 Culling Surface Primitive

In principle, assuming that the input model $\boldsymbol{\Gamma}$ is triangulated, each $SV(f_k^{\boldsymbol{\Gamma}})$ generates three ruled surface primitives, one developable surface, and $f_k^{\boldsymbol{\Gamma}}$'s at the initial and final positions of $\boldsymbol{\tau}(t)$. Therefore, the triangle counts of the ruled and developable surfaces significantly affect the performance of the pipeline presented in Fig. 2. Consequently, we want to identify portions of surface primitives that do not contribute to $\partial SV(\boldsymbol{\Gamma})$, and prune them away accordingly. We use a variation of a technique presented in [8] to cull away redundant ruled surface primitives, and also provide a novel method for developable surface primitives.

In order to prune ruled surface primitives, we perform the following operation. First of all, a *reflex edge* e_r in $\boldsymbol{\Gamma}$ is not used to generate a ruled surface at all, since the surface will be always subsumed by the SV of the adjacent faces of e_r (also see Fig. 4-(a)). The same reasoning is applied to a *coplanar edge*, whose adjacent faces are coplanar. Furthermore, if a *convex edge* e_c instantaneously moves inward $f_l^{\boldsymbol{\Gamma}}$ and $f_m^{\boldsymbol{\Gamma}}$ at time t, where $f_l^{\boldsymbol{\Gamma}}$ and $f_m^{\boldsymbol{\Gamma}}$ are the adjacent faces of e_c, then e_c can stop generating a ruled surface at that time, since that portion will be also subsumed by $SV(f_l^{\boldsymbol{\Gamma}})$ or $SV(f_m^{\boldsymbol{\Gamma}})$ (also see Fig. 4-(b)). This test can be easily worked out by checking the velocity vectors $\boldsymbol{\tau'}(t)$ at the endpoints of e_c against the face normals of $f_l^{\boldsymbol{\Gamma}}$ and $f_m^{\boldsymbol{\Gamma}}$.

We also present a novel culling scheme for developable surface primitives. The main idea is that we generate a



(a) (b) (c)

Figure 4: Surface Culling. ***(a) A reflex edge e_r is not needed to generate a ruled surface along the trajectory τ, because the surface will be subsequently subsumed by the SV of its adjacent faces, $SV(f_l)$ or $SV(f_m)$. (b) A convex edge e_c does not need to produce a ruled surface when it is swept inside of its adjacent faces (f_l and f_m) along τ, because it will be subsumed by $SV(f_l)$ or $SV(f_m)$. (c) A developable surface d does not need to be created when it exists inside its generator triangle. This is checked by the angle between the normal n_t and the difference vector $\boldsymbol{d}(t+\Delta t,v)-\boldsymbol{d}(t,v)$ between successive time-steps.***

developable surface $\boldsymbol{d}_{f_k^{\boldsymbol{\Gamma}}}(t,v)$ only if its boundary can be exposed outside of its generating face $f_k^{\boldsymbol{\Gamma}}$. Since a developable surface $\boldsymbol{d}_{f_k^{\boldsymbol{\Gamma}}}(t,v)$ is locally convex [18] and $f_k^{\boldsymbol{\Gamma}}$ is always tangent to $\boldsymbol{d}_{f_k^{\boldsymbol{\Gamma}}}(t,v)$, $\boldsymbol{d}_{f_k^{\boldsymbol{\Gamma}}}(t,v)$ locally lies inside or outside of $f_k^{\boldsymbol{\Gamma}}$ depending on the face normal $\boldsymbol{n}_t$ of $f_k^{\boldsymbol{\Gamma}}$. More specifically, since we perform uniform tessellation of a developable surface using some fixed time step Δt, we approximate the locality with Δt. Then, we compute two points $\boldsymbol{d}_{f_k^{\boldsymbol{\Gamma}}}(t,v)$ and $\boldsymbol{d}_{f_k^{\boldsymbol{\Gamma}}}(t+\Delta t,v)$ from Eq. 5, and check the angle between the difference vector $\boldsymbol{d}_{f_k^{\boldsymbol{\Gamma}}}(t+\Delta t,v)-\boldsymbol{d}_{f_k^{\boldsymbol{\Gamma}}}(t,v)$ and the plane normal $\boldsymbol{n}_t$ of $f_k^{\boldsymbol{\Gamma}}$. If it is less than 90 degrees, $f_k^{\boldsymbol{\Gamma}}$ is used during time t, otherwise it is pruned away (also see Fig. 4-(c)).

5. SAMPLING AND RECONSTRUCTION

Once we have computed all the surface primitives of SV, we approximate the outer boundary of SV by sampling the surfaces and reconstructing the outer boundary of their arrangement. In this section, we describe the sampling and reconstruction pipeline (see Fig. 2). We compute an unsigned distance field with respect to the surface primitives on a discrete spatial grid. A signed distance field is obtained by propagating a front around the boundary of the swept volume using a fast marching method. An iso-surface extraction from this signed distance field provides us with an initial approximation to the outer boundary. We perform a topological connectedness test on this approximation. If the test fails, we refine the spatial grid, recompute the distance field, and repeat the pipeline.

5.1 Distance Field Representation

Given all the surface primitives of SV, we first discretize the 3D space occupied by the primitives. As a discrete representation of the 3D space, we choose signed distance fields with respect to the surface primitives, and attempt to compute them efficiently using graphics hardware. This discrete representation is used later in iso-surface extraction.

We sample the distance values at the discrete points of a 3D spatial grid, and use an enhanced representation of the discrete distance field. Here, the distance value at each grid point means the closest distance to one of the surface primitives. However, in our scheme, instead of simply using a scalar distance value for each grid point, we store directed distances along six principal directions corresponding to $x-$, $x+$, $y-$, $y+$, $z-$ and $z+$ axes. Our goal is to evaluate the directed distance function at the grid points of a 3D uniform grid. We would like to use an approach that maps well to SIMD-like capabilities of rasterization hardware. Current graphics processors have the capability to evaluate the distance function in parallel for each pixel on the plane. Graphics hardware-based fast techniques have been used for distance field evaluation [24].

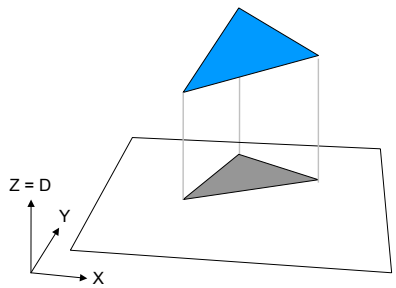


Figure 5: Directed distance field: ***This figure shows how a slice of the directed distance field of a primitive (blue triangle) is computed. The primitive is rendered under orthographic projection with the slice (black rectangle) set as the image plane. The Z-buffer holds the directed distance values. The grey triangle is the projection of the primitive onto the slice.***

We employ a modified approach, and also use graphics hardware to generate the directed distance fields as follows:

1. Our algorithm computes the directed distance along a given direction by sweeping a plane along that direction. This plane corresponds to a slice of the directed distance field and is perpendicular to the direction of sweep.

2. Our algorithm computes the directed distance field one slice at a time. So the problem is reduced to defining the directed distance function of a primitive over a planar 2D slice. The main idea of our approach is that in order to obtain an approximation to the primitive's directed distance function, we simply render the primitive under orthographic projection with the slice as the image plane (see Fig. 5).

3. At each step, the slice is moved by a distance equal to the size of the grid cell. The planes corresponding to two consecutive slices are used to define a slab.

4. For each slab, we precompute the set of surface primitives that it intersects with.

5. We use orthographic projection to sample and rasterize the surfaces. The above slab is set as the near and far clipping planes. We render the surface primitives associated with the slab. Each pixel in the frame buffer corresponds to a point in the current slice and the depth buffer holds the value of the distance at that point.

6. We readback the depth buffer and store the directed distance values. Distances with absolute values larger than grid edge length are irrelevant since they are not used during isosurface extraction.

Our algorithm computes only an unsigned directed distance field. However, the isosurface extraction algorithm requires a signed distance field; i.e., a distinction needs to be made between inside and outside.

5.2 Fast Marching Front Propagation

We perform an inside/outside classification at each grid point to obtain a signed distance field. Conventionally, points that lie outside the boundary of the SV have a positive sign while those inside have a negative sign. Our surface primitives are in general not closed. As a result, we cannot define an inside/outside classification with respect to the individual surface primitives. We need to define a classification with respect to the boundary of the SV. However, this classification problem is non-trivial because we do not know the boundary of the SV.

In order to solve the inside and outside classification problem, we present a variant of the fast marching level set method [43] to propagate a front around the boundary of the swept volume. Level-set methods are numerical techniques for computing the position of propagating fronts. Topological changes are naturally captured in this setting. We perform the front propagation on the discrete spatial grid (see Fig. 6). We use the unsigned directed distance field generated in Sec. 5.1 for front propagation. The front consists of a set of grid points. We can initialize the front to be a set of grid points corresponding to any surface bounding the SV. One choice for the initial front is the set of grid points that lie along the boundary of the spatial grid. Our front propagation method ensures that the front visits exactly those grid points that lie outside the swept volume.

We tag grid points as *Known*, *Trial*, or *Far* depending on whether the grid point has already been visited, is currently being visited, or is yet to be visited by the front, respectively. Each grid point also has a flag whose value can be *Inside* or *Outside*. Initially all grid points are assigned a flag, *Inside*. All grid points except the initial front are tagged as *Far*. Grid points on the initial front are tagged as *Trial*. During one step of front propagation, we perform a number of operations. These include:

1. We arbitrarily pick a *Trial* grid point belonging to the front and remove it from the front. Let this point be denoted as P. We set its tag to be *Known*.

2. Consider a neighboring grid point Q of P. If point Q is tagged as *Known*, we do not update it. With respect to P, point Q lies along one of the six principal directions. We check if the directed distance of P along that direction is larger than the length of edge connecting P and Q. If that is the case, we are guaranteed that point Q lies outside the boundary of the SV. Therefore we propagate the front to point Q by adding Q to the front. In addition, the flag for point Q is set to *Outside*.

The pseudo-code is shown in Alg. 5.1. The front propagation continues in this manner until the front has visited all grid points outside the SV. At this time, front propagation terminates. In this manner, we obtain an inside/outside classification for each grid point. We combine this inside/outside classification with the unsigned distance field computed in Sec. 5.1 to obtain a signed distance field. All six directed distances at a grid point always have the same sign.

```
while front is nonempty
  Extract a trial point P from the front
  P.tag = KNOWN
  for each neighbor Q of P,
    if Q.tag != Known then
      d = Direction from P to Q
      if Directed_Distance(P,d) > Edge_Length(P,Q) then
        Q.flag = Outside
        if Q.tag == Far then
          Add Q to the Front
          Q.tag = Trial
        endif
      endif
    endif
  endfor
endwhile
```

ALGORITHM 5.1: Fast Marching Method

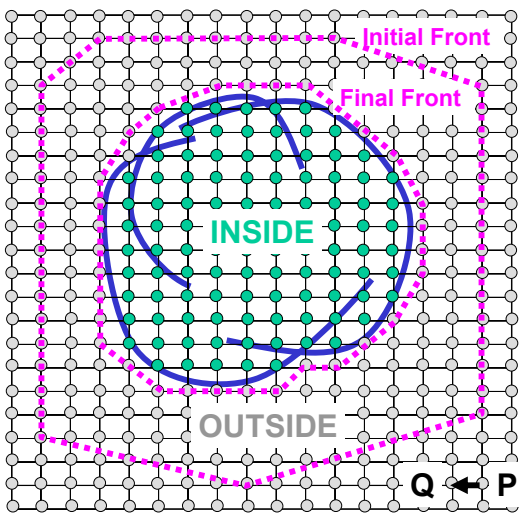


Figure 6: Fast Marching Method. ***A fast marching level set method is used to propagate a front (pink dotted curve) around the surface primitives of SV (solid blue curves). All the grid points visited by the front (grey circles) lie outside the outer boundary while the remaining grid points (green circles) lie inside the outer boundary. During front propagation, a grid point P can update its neighboring grid point Q if the directed distance from P to Q is greater than the length of the edge connecting P and Q***

5.3 Isosurface Extraction

We estimate the outer boundary of open surfaces by performing an isosurface extraction from the signed distance field generated using the approach described in Sec. 5.1 and Sec. 5.2. We use the Extended Marching Cubes (EMC) algorithm [32] to perform the isosurface extraction. This algorithm can detect sharp features and sample them in order to reduce the aliasing artifacts. The output of the isosurface

extraction is a polygonal mesh. This is our initial approximation to the outer boundary.

In order to perform isosurface extraction, we need to know which edges of a cube of the spatial grid are intersected by the isosurface. An edge of a cube is intersecting if the two endpoints of the edge have different inside/outside classification. For each intersecting edge of a cube, the directed distances of the endpoints of the edge give us the position of the intersection point (see Fig. 7). The advantage of using directed distance is that it provides us with exact surface samples. The standard Marching Cubes algorithm can produce aliasing artifacts in the vicinity of sharp features. The Extended Marching Cubes algorithm uses a tangent element approximation to reduce aliasing artifacts and provide better reconstruction in the presence of sharp features. Thus we have a better approximation to the exact, outer boundary.

We only use the directed distance of the grid point which is *Outside*. The directed distance of the grid point which is *Inside* may result in incorrect intersection points (see Fig. 7).

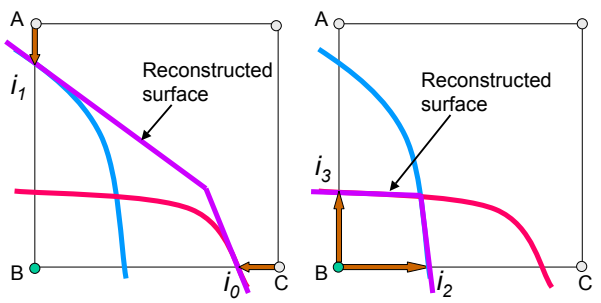


Figure 7: Iso-Surface Extraction: ***Figure on the left shows the reconstructed surface (purple) for two surface primitives (blue and red). We use the directed distance (show in brown) to compute intersection points (i0 and i1). The grey and green circles respectively indicate grid points that lie Outside and Inside the outer boundary. Figure on the right shows that the directed distance of a Inside grid point (Point B) may result in incorrect intersection points (i2 and i3). We only use the directed distance of Outside grid points for reconstruction.***

5.4 Topological Refinement

The underlying topology induced by our SV approximation algorithm can be different from the topology of the exact SV. This mainly results from the sampling and reconstruction steps in our computational pipeline. However, our algorithm attempts to maintain some of the topological properties of SV, i.e. closed and connected boundary. According to our sweep equation in Eq. 1, the SV that we generate for a polyhedron can be a non-manifold. However, its boundary always provides a closed, water-tight surface, since the generator is a closed set. Moreover, the structure always generates one connected component.

To ensure a single connected component in our SV algorithm, we perform a topological check by traversing the generated polygonal mesh to detect the occurrence of such a case (see Fig. 8) . We arbitrarily pick a vertex from the mesh. We mark this vertex and recurse on each of the unmarked neighboring vertices. At the end of this traversal, if any unmarked vertices remain, it implies that the mesh has more than one component. In that case, we refine the spatial grid, recompute the distance field at a higher resolution, and perform the reconstruction again. We use EMC algorithm for the iso-surface reconstruction. This algorithm always generates a closed polygonal mesh structure. Therefore, our SV algorithm can guarantee closed, connected surface structures.

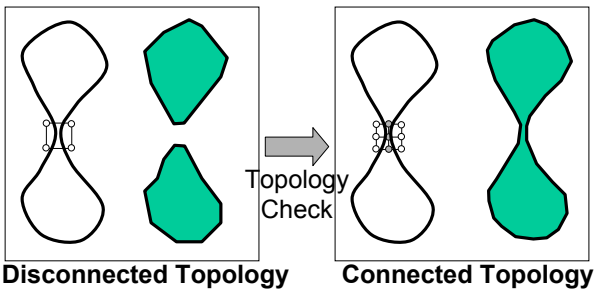


Figure 8: Topological Refinement. ***We ensure that our SV approximation is a single connected component by performing topological refinement. We perform a topological check to see if our approximation has more than one component. In that case, we refine the spatial grid and perform the reconstruction again.***

6. ANALYSIS OF SWEPT VOLUME ALGORITHM

In this section, we analyze the performance of our SV algorithm, and also discuss the sources of errors in the algorithm.

6.1 Performance Analysis

Our SV algorithm has the following computational complexity:

Surface Generation: The computational cost of the surface generation is mainly determined by the surface tessellation process, and its complexity is sensitive to the output; i.e. $O(M)$, where M is the number of triangles generated by the tessellation. Let us denote N_e as the total number of convex edges of the input polyhedra $\mathbf{\Gamma}$, and N_f as the total number of faces of $\mathbf{\Gamma}$. Assuming that $\mathbf{\Gamma}$ is triangulated, in the worst case, $3N_e$ ruled surfaces and N_f developable surfaces are generated. Let us further denote $M_e^i(\epsilon, \boldsymbol{\tau})$ as the number of triangles generated by a ruled surface i, which depends on the given surface deviation error ϵ and the trajectory $\boldsymbol{\tau}$, and denote $M_f^j(\epsilon, \boldsymbol{\tau})$ as the number of triangles by a developable surface j. Let $M_e^+ = \max_{i=1}^{3N_e}(M_e^i(\epsilon, \boldsymbol{\tau}))$ and $M_f^+ = \max_{j=1}^{N_f}(M_f^j(\epsilon, \boldsymbol{\tau}))$. Then, $M = 3N_e M_e^+ + N_f M_f^+$. Typically, in our experiments, M_e^+ and M_f^+ correspond to a few hundred triangles.

Distance Field Generation: Let D be the maximum dimension of the bounding box enclosing the surface primitives of SV. Then, let $K = D/\epsilon$, where ϵ is the given surface deviation error. We use a $K \times K \times K$ uniform spatial grid for generating the distance field, front propagation and isosurface extraction to restrict the reconstruction error within ϵ (see Sec. 6.2 for more detail). As stated earlier, we compute the distance field using graphics hardware. We can measure the time complexity of the distance field generation in terms

of number of primitives sent to the graphics hardware. A primitive p_i gets rendered n_{p_i} times where N is the number of primitives and n_{p_i} is the number of slabs it occupies. Also the size of the spatial grid contributes to the time complexity. So the total time complexity is $O(K^3) + O(\sum_{i=1}^{N} n_{p_i})$. Typically, n_{p_i} is a small constant for most primitives. So the time to generate directed distance fields is typically linear in number of primitives.

Fast Marching Front Propagation: Front propagation takes time proportional to the size of the spatial grid; i.e., $O(K^3)$.

Isosurface Extraction: Isosurface extraction takes time proportional to the size of the spatial grid; i.e., $O(K^3)$.

Total Complexity: The total computational complexity of our SV algorithm is $O(M + K^3 + \sum_{i=1}^{N} n_{p_i})$.

6.2 Error Analysis

Our SV algorithm is an approximation scheme. There are three main sources of the errors that govern the accuracy of the result of our algorithm; tessellation errors by approximating surface primitives, sampling errors from generating 3D grids of distance fields, and iso-surface reconstruction errors from the EMC.

Tessellation Errors: We use adaptive tessellation to triangulate ruled surfaces, and uniform tessellation to triangulate developable surfaces. These methods can triangulate the surface primitives within an error threshold ϵ, which is essentially the Hausdorff distance between the original surfaces and approximated ones.

Sampling Errors: The accuracy of the distance field is dependent on the implementation. We compute it using graphics hardware and its accuracy is determined by the number of bits of precision in the Z-buffer, typically 24 or 32 bits in current hardware.

Reconstruction Errors: If ϵ is the size of the grid cell, we are guaranteed that each point on our reconstructed outer boundary lies within a distance ϵ of some point on the exact envelope. The approximation theory guarantees that a piecewise linear interpolant to a smooth surface converges with order $O(\epsilon^2)$ where ϵ measures the sampling density. In our case, ϵ is the size of the grid cell. In the presence of sharp features, however, the convergence rate drops to $O(\epsilon)$. However, the Extended Marching Cubes algorithm improves the local convergence rate by performing a tangent element approximation. This convergence rate is valid only in cells that have at most one sharp feature.

7. IMPLEMENTATION AND PERFORMANCE

In this section, we describe the implementation of our SV algorithm and highlight its performance on different benchmarks.

7.1 Implementation

To implement our SV algorithm, we used C++ programming language with the GNU g++ compiler under Linux operating system. For the choice of GUI implementation, GLUT, OpenGL, Inventor and Qt were used.

We used a public computational geometry library, CGAL, to perform an efficient traversal on the two-manifold polyhedral surfaces. Moreover, CGAL offers quite flexible data structures based on the usage of templates and STL programming, and also provide accurate evaluation of geometric predicates such as orientation test, cross product, dot product, etc [20]. We took advantage of these benefits to implement the generation of surface primitives of SV. In particular, the *Polyhedron_3* class of CGAL was extensively used.

In order to compute the distances fields quickly and efficiently, we used nVidia's GeForce 4 GPU, which has 24 bit precision of accuracy in Z-buffer. With the availability of new GPU's such as ATI's Radeon9700, we can further improve this accuracy by using their floating point computational capability in the graphics pipeline.

7.2 Performance

We benchmarked our SV algorithm by using different models of varying complexities and with different sweeping trajectories. The complexity of our benchmarking models varies from 1,524 to 10,352 triangles. The model complexities are summarized from the second to the fifth column in Table 1. Furthermore, they consist of sharp edges and surface triangles with high aspect ratio. The sweeping paths that we chose are helical sweep (X-Wing and Swing-Clamps), translations using cubic rational functions (Input Clutch), and sinusoidal translations and rotations (the rest of the models). Therefore, most of our benchmarks perform sweep along very general trajectories. For the grid resolution in our benchmarks, we use the grid resolution $K = 128$ for all the models.

We performed timing analysis on a PC with Intel Xeon 2.4 GHz processor, 2GB of memory and nVidia GeForce 4 graphics card. The time spent during each stage in our SV computational pipeline is shown in different columns of Table 1. As the table shows, most of the time, typically more than 80% of the total computational time, is spent in the distance field generation stage of the pipeline. We observed that the distance field computation time was mainly spent on the readbacks between the framebuffer and main memory. Thus, as we increase the grid size K, the total computational time will increase linearly, since we perform the readbacks $O(K)$ times. We measured performance of our SV computation pipeline on the hammer benchmark at a grid resolution of $K = 256$. The distance field computation, front propagation and isosurface reconstruction took 41.4 s, 12.6 s and 8.9 s respectively.

In Fig. 9, we illustrate the results of the SV of the benchmarking models computed by our SV algorithms. In the figure, each row shows the generator polyhedral model $\mathbf{\Gamma}$, sweeping path $\boldsymbol{\tau}$, and the resulting SV approximation $\partial SV(\mathbf{\Gamma})$, respectively. All the rendered images in Fig. 9 are flat-shaded.

8. SUMMARY AND FUTURE WORK

We present an efficient, fast algorithm to approximate SV of complex polyhedral models using the distance fields, fast marching propagation method, and iso-surface reconstruction. The algorithm has been benchmarked on a number of complex models with different sweep paths.

Model	Combinatorial Complexity				Computational Performance (seconds)				
	$\mathbf{\Gamma}$	# of Surf	# of Surf Tri	$\partial SV(\mathbf{\Gamma})$	Surf Gen	Dist Field	Front Prop	Isosurface	Tot
X-wing	2496	3931	770K	307K	3.208	36.15	1.69	3.12	44.1
Air Cylinder	2280	1152	234K	249K	1.966	16.0	1.65	1.55	21.16
Swing Clamps	1524	1049	212K	126K	1.492	15.7	1.73	1.33	20.2
Hammer	1692	1390	281K	198K	1.822	16.1	1.59	1.97	21.4
Input Clutch	2116	1175	239K	129K	1.789	16.2	1.61	1.39	20.9
Pipe	10352	15554	803K	247K	4.038	59.2	1.61	2.48	67.2
Pivoting Arms	2158	1718	347K	162K	2.138	21.4	1.60	1.64	26.7

Table 1: Model Complexities of SV Benchmarks. ***The first column shows the model names of the benchmarks. From the second to the fifth column, each column respectively shows the triangle count of the generator $\mathbf{\Gamma}$, the number of ruled and developable surfaces, the total number of triangles in the tessellated ruled and developable surfaces, and the triangle count of the boundary of $\partial SV(\mathbf{\Gamma})$ computed by our algorithm. From the sixth to the tenth column, each column respectively illustrates the timing for the surface primitive generation, distance field generation, inside/outside classification using fast marching propagation, iso-surface extraction using the EMC, and the total computation. We have chosen a grid resolution of $K = 128$ for all the benchmarks.***

There are several areas for future work. We would like to look at adaptive subdivision schemes for better reconstruction [45]. The performance of our algorithm can be further improved by investigating more optimizations. These include more efficient surface generation based on incremental computations on sweeping path, possibility of using programmable graphics hardware to simulate the fast marching method, etc. We would like to further investigate the application of our SV algorithm to the areas such as collision detection, robot workspace analysis, and computer-aided geometric design. Finally, we will like to extend this algorithm to solids bounded by curved surfaces.

9. ACKNOWLEDGEMENTS

This research is supported in part by ARO Contract DAAD 19-99-1-0162, NSF awards ACI-9876914, IIS-982167, ACI-0118743, ONR Contracts N00014-01-1-0067 and N00014-01-1-0496, and Intel Corporation. We thank Kenny Hoff for his HAVOC software, Leif Kobbelt for providing us with his Extended Marching Cubes software, and Dan Halperin and the reviewers for their feedback and suggestions.

10. REFERENCES

[1] K. Abdel-Malek, D. Blackmore, and K. Joy. Swept volumes: Foundations, perspectives, and applications. *International Journal of Shape Modeling*, 2002. submitted.

[2] K. Abdel-Malek and S. Othman. Multiple sweeping using the denavit-hartenber representation method. *Computer-Aided Design*, 31:567–583, 1999.

[3] K. Abdel-Malek, J. Yang, R. Brand, M. Vannier, and E. Tanbour. Towards understanding the workspace of human limbs. *International Journal of Ergonomics*, 2002. submitted.

[4] K. Abdel-Malek and H. J. Yeh. Analytical boundary of the workspace for general 3-DOF mechanisms. *International Journal of Robotics Research*, 16:1–12, 1997.

[5] K. Abdel-Malek and H. J. Yeh. Geometric representation of the swept volume using the jacobian rank deficiency conditions. *Computer-Aided Design*, 29(6):457–468, 1997.

[6] K. Abdel-Malek and H. J. Yeh. On the determination of starting points for parametric surface intersections. *Computer-Aided Design*, 29(1):21–35, 1997.

[7] S. Abrams and P. Allen. Swept volumes and their use in viewpoint computation in robot work-cells. In *Proc. IEEE International Symposium on Assembly and Task Planning*, pages 188–193, 1995.

[8] S. Abrams and P. Allen. Computing swept volumes. *Journal of Visualization and Computer Animation*, 11, 2000.

[9] J.-W. Ahn, M.-S. Kim, and S.-B. Lim. Approximate general sweep boundary of a 2D curved objects. *Graphical Models and Image Processing*, 55(2):98–128, 1993.

[10] N. Baek, S. Shin, and K. Chwa. Three-dimensional topological sweep for computing rotational swept volumes of polyhedral objects. *Int'l J. of Computational Geometry and Applications*, 10(2), 2000.

[11] D. Blackmore, M. Leu, and L. Wang. Sweep-envelope differential equation algorithm and its application to NC machining verification. *Computer-Aided Design*, 29:629–637, 1997.

[12] D. Blackmore and M. C. Leu. A differential equation approach to swept volumes. In *Proc. of Rensselaer's 2nd International Conference on Computer Integrated Manufacturing*, pages 143–149, 1990.

[13] S. Boussac and A. Crosnier. Swept volumes generated from deformable objects application to NC verification. In *Proceedings of International Conference on Robotics and Automation*, pages 1813–1818, Apr 1996.

[14] A. J. Chung and A. J. Field. A simple recursive tessellator for adaptive surface triangulation. *Journal of graphics tools*, 5(3):1–9, 2000.

[15] D. Cohen-Or, A. Solomovic, and D. Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, 1998.

[16] J. Conkey and K. Joy. Using isosurface methods for visualizing the envelope of a swept trivariate solid. In *Proc. of Pacific Graphics*, Oct 2000.

[17] M. de Berg, L. J. Guibas, and D. Halperin. Vertical decompositions for triangles in 3-space. *Discrete and Computational Geometry*, 15:36–61, 1996.

[18] M. de Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, Englewood Cliffs, NJ, 1976.

[19] G. Elber and M.-S. Kim. Offsets, sweeps, and Minkowski sums. *Computer-Aided Design*, 31(3):163, 1999.

[20] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. The CGAL kernel: A basis for geometric computation. In M. C. Lin and D. Manocha, editors, *Proc. 1st ACM Workshop on Appl. Comput. Geom.*, volume 1148, pages 191–202. Springer-Verlag, 1996.

[21] S. Frisken, R. Perry, A. Rockwood, and T. Jones. Adaptively sampled distance fields: A general representation of shapes for computer graphics. *Proc. of ACM SIGGRAPH*, pages 249–254, 2000.

[22] S. F. F. Gibson. Using distance maps for accurate surface representation in sampled volumes. In *IEEE Symposium on Volume Visualization*, pages 23–30, 1998.

[23] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.

[24] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of ACM SIGGRAPH*, pages 277–286, 1999.

[25] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast and simple geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics*, 2001.

[26] Y. Huang and J. H. Oliver. NC milling error assessment and tool path correction. In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 287–294. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[27] K. Hui. Solid sweeping in image space - application to NC simulation. *Visual Computer*, 1994.

[28] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. In *SIGGRAPH 2002, Computer Graphics Proceedings*, 2002.

[29] B. Jüttler and M. Wagner. Spatial rational B-spline motions. *ASME Journal of Mechanical Design*, 118:193–201, 1996.

[30] J. Kieffer and F. Litvin. Swept volume determination and interference detection for moving 3-D solids. *ASME Journal of Mechanical Design*, 113:456–463, 1990.

[31] R. Kimmel, N. Kiryati, and A. Bruckstein. Multivalued distance maps for motion planning on surfaces with moving obstacles, 1998.

[32] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature-sensitive surface extraction from volume data. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 57–66. ACM Press / ACM SIGGRAPH, 2001.

[33] S. Kumar and D. Manocha. Efficient rendering of trimmed nurbs surfaces. *Computer-Aided Design*, pages 509–521, 1995.

[34] C. Law, S. Avila, and W. Schroeder. Application of path planning and visualization for industrial design and maintainability-analysis. In *Proc. of the 1998 Reliability and Maintainability Symposium*, pages 126–131, 1998.

[35] J. Lee, S. Hong, and M. Kim. Polygonal boundary approximation for a 2D general sweep based on envelope and boolean operations. *Visual Computer*, 16, 2002.

[36] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, 1987.

[37] C. Madrigal and K. Joy. Boundary determination for trivariate solids. In *Proc. of the IASTED Intl Conf on Computer Graphics and Imaging*, Oct 1999.

[38] R. Martin and P. Stephenson. Sweeping of three-dimensional objects. *Computer-Aided Design*, 22(4), 1990.

[39] H. Pottmann and J. Wallner. *Computational Line Geometry*. Springer, 2001.

[40] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th ACM Symposium on Computational Geometry*, pages 163–172, 1999.

[41] W. Schroeder, W. Lorensen, and S. Linthicum. Implicit modeling of swept surfaces and volumes. In *IEEE Visualization Conference*, 1994.

[42] F. Schwarzer, M. Saha, and J.-C. Latombe. Exact collision checking of robot paths. In *International Workshop on Algorithmic Foundations of Robotics*, 2002.

[43] J. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proc. Nat. Acad. Sci.*, volume 93, pages 1591–1595, 1996.

[44] T. Van Hook. Real-time shaded NC milling display. In D. C. Evans and R. J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 15–20, Aug. 1986.

[45] G. Varadhan, S. Krishnan, Y. Kim, and D. Manocha. Adaptive subdivision and reconstruction using box-distance fields. *Technical Report TR03-005, Department of Computer Science, University of North Carolina*, 2003.

[46] L. Velho, L. H. de Figueiredo, and J. Gomes. A unified approach for hierarchical adaptive tesselation of surfaces. *ACM Transactions on Graphics*, 18(4):329–360, 1999.

[47] W. Wang and K. Wang. Real-time verification of multi-axis NC programs with raster graphics. In *Proceedings of International Conference on Robotics and Automation*, pages 166–171, 1986.

[48] J. Weld and M. Leu. Geometric representation of swept volume with application to polyhedral objects. *International Journal of Robotics Research*, 9(5), 1990.

[49] A. Winter and M. Chen. Image-swept volumes. In *Proc. of Eurographics*, 2002.

[50] Z. Wood, M. Desbrun, P. Schroeder, and D. Breen. Semi-regular mesh extraction from volumes. In *IEEE Visualization 2000 Proceedings*, 2000.

[51] P. Xavier. Fast swept-volume distance for robust collision detection. In *Proceedings of International Conference on Robotics and Automation*, 1997.

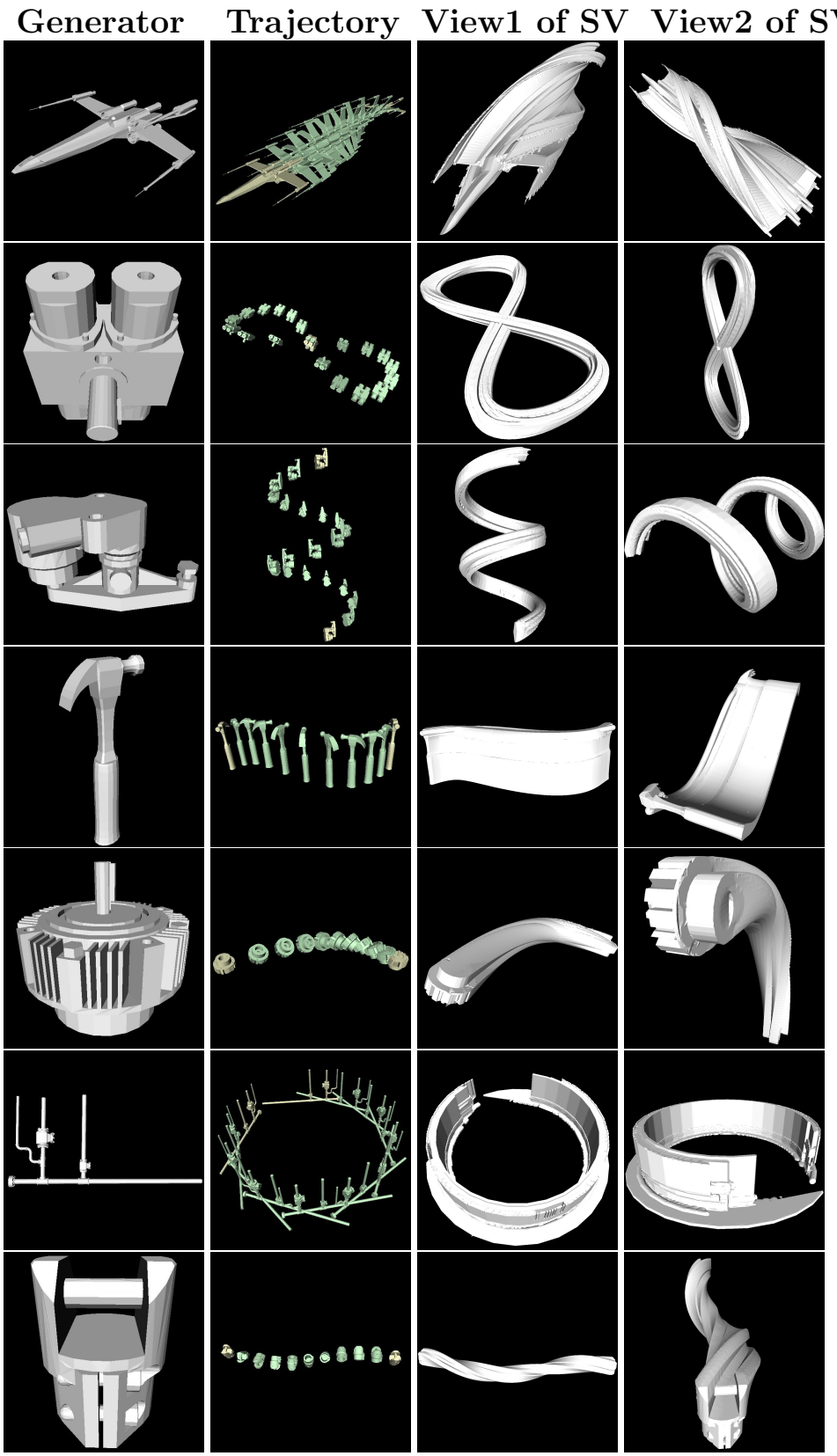


Figure 9: SV Benchmarks. *In each column, from left to right, each figure shows a generator model, sweeping trajectory, and two views of the resulting SV approximation reconstructed by our SV algorithm, respectively. In each row, each figure shows different benchmarking model, from top to bottom, X-Wing, Air Cylinder, Swing Clamps, Hammer, Input Clutch, Pipe, and Pivoting Arms, respectively.*