THE JOURNAL OF VISUALIZATION AND COMPUTER ANIMATION
J. Visual. Comput. Animat. 2000; **11**: 69–82

Computing swept volumes

By Steven Abrams† and Peter K. Allen*

The swept volume problem is practical, difficult and interesting enough to have received a great deal of attention over the years, and the literature contains much discussion of methods for computing swept volumes in many situations. The method presented here permits an arbitrary polyhedral object (given in a typical boundary representation) to be swept through an arbitrary trajectory. A polyhedral approximation to the volume swept by this moving object is computed and output in a typical boundary representation. A number of examples are presented demonstrating the practicality of this method.

Received: 15 February 1999; Revised: 15 November 1999

Introduction

Swept volumes have been used in a wide variety of applications. While useful in such varied contexts as designing the core and cavity of injection-moulded and die-cast components[1] and robot workspace generation,[2] most applications relate to virtual simulations of real-world motions for one purpose or another. These include collision avoidance,[3] NC tool path verification,[4–6] computing the 'removal envelopes' of mechanical parts (to facilitate maintainability design in complex mechanical systems)[7] and virtual assembly where articulated parts can be virtually prototyped and assembly plans tested and modified without resorting to physically building objects.[8] As it is one of the topics within computational geometry which is at once interesting, practical and challenging, it has received quite a bit of investigation over the years.

This paper presents a new method for computing the volumes swept by moving polyhedral objects as they travel through arbitrary trajectories in space. Our interest in swept volumes came from our ongoing research in sensor planning. We have been looking at methods of computing viewpoints which allow the monitoring of specified object features while other objects in the environment are moving.[9–11] We have developed a method based on temporal intervals in which a portion of the task is selected to be monitored. The system computes the volumes swept by all moving objects during this interval and computes viewpoints which avoid occlusion by these swept volumes. Such viewpoints would be valid for the entire task interval. By examining appropriate time intervals, we break the *dynamic sensor planning* problem down into a series of static problems. Therefore the swept volume computation is central to our sensor planning work. Our initial swept volume efforts were reported in Reference 12. This paper presents a more robust and efficient algorithm than our earlier work, along with a more thorough discussion of the underlying theory.

*Correspondence to: S. Abrams, IBM T. J. Watson Research Center, Department of Mathematical Sciences, Yorktown Heights, NY 10598, USA. E-mail: abrams@watson.ibm.com
† The research described in this paper was performed while this author was at the Columbia University Department of Computer Science.

Swept Volume Definitions

A 'swept volume' is a fairly intuitive concept. Nevertheless, it is important to establish a formal definition so as to clarify exactly what we will be computing. Let $\mathrm{M} \subset \mathbf{R}^d$ be the object to be swept through $\mathbf{R}^d$. Let H be a $(d+1)$-dimensional transformation matrix defining a position and orientation in $\mathbf{R}^d$. Such matrices are of the form

$$H = \begin{bmatrix} R & T \\ 0 \ldots 0 & 1 \end{bmatrix} \tag{1}$$

where R is a d-dimensional orthonormal rotation matrix and T is a $d \times 1$ translation vector.

We can now define M_H to be M transformed by the transformation matrix H, i.e.

$$M_H = \{Hm : m \in M\} \tag{2}$$

The trajectory of M as it moves can be described as a time-varying transformation matrix $Q=H(t)$ defined over a time interval $[t_0, t_1]$. Q yields a new position and orientation at every time t.

We can now define the sweep of $M \subset \mathbf{R}^d$ (often called the *generator*) over an arbitrary trajectory Q (notated $\mathscr{S}(M, Q)$ as

$$\mathscr{S}(M,Q) = \bigcup_{t\in[t_0,t_1]} M_H(t) \tag{3}$$

Intuitively, we are dragging M through a path, possibly twisting and turning it as we go.

The cases which will be most interesting to us are those in which the transformations $H(t)$ change continuously over time, since that is the way in which objects move—continuously. The formal way of stating what we mean by 'moving continuously' is

$$\forall m\in M,\ \lim_{\Delta t\to 0} D(H(t)m, H(t+\Delta t)m) = 0 \tag{4}$$

where $D(a, b)$ is the Euclidean distance metric in $\mathbf{R}^d$.

Because of the nature of the analysis which we will be doing for sensor planning, we will be concerned with computing a boundary representation of a polyhedral approximation to the volume swept by M.

Related Sweeping Research

As we have mentioned, the swept volume problem has received quite a bit of investigation over the years. Here we review a representative set of the literature, beginning with approaches that handle only translational motion of polyhedra, and progressing to work which handles arbitrary motions of non-rigid bodies.

Translational Sweeps

The sweep of an object A as it moves along a purely translational trajectory B is the Minkowski sum of A and B (written $A \oplus B$). A Minkowski sum is a vector sum defined as

$$A \oplus B \equiv \{a+b | a\in A, b\in B\} \tag{5}$$

The Minkowsi sum is useful for 'growing' objects, since if, for example, B is a ball of radius r, $A \oplus B$ is A 'grown' by r. Kaul[13] has developed algorithms for computing the Minkowski sums of polyhedral objects efficiently and robustly. One of the component steps in computing these Minkowski sums is sweeping the polyhedron along the edge of a polygon. Therefore his methods can be applied to computing the volumes swept by the *piecewise* linear translational motion of polygons. Translational sweeps of non-polyhedral objects can be computed using a polyhedral approximation.

A very different method of computing the translational sweeps of objects is presented in Reference 14. Here the generator is represented using an array, similar in concept to a Z-buffer. Each entry in the array contains a list of depth elements or 'dexels' representing the range of depths containing the object at that pixel location. This is a form of an image space representation of the object. Hui[14] computes piecewise linear translational sweeps directly in this image space by computing the straight lines swept by each dexel using a 3D scan conversion process. These dexels are merged using Boolean operations in image space, yielding a dexel representation of the swept object. They apply their sweeping methods to the visual verification of NC tool paths.

Rotations

In one of the earlier applications of swept volumes to robotics problems, Korein uses swept volumes to compute robot workspace bounds.[2] He computes a polyhedral approximation to the volume swept by a polyhedron rotating about a single axis or radially rotating about a point. To accomplish this, he examines extreme features with respect to the axis of rotation and connects them with a piecewise planar approximation to the curved surface patch which would result from sweeping these features. This yields a superset of the boundary elements of the swept volume. The members of this superset are intersected with each other, and the outermost boundary is found by using a graph traversal algorithm. Korein's methods can therefore be used to compute approximate rotational sweeps of polyhedra. Though limited to rotational sweeps, we will later show how this basic technique can be generalized to arbitrary trajectories.

General Three-dimensional Motion

A number of researchers have used *envelope theory* for studying or computing swept volumes. Wang and Wang[4] examine the volumes swept by convex generators, considering each moving boundary of the generator as defining a family of surfaces, and discuss the computation of the envelope of this family to bound the swept volume. For convex generators,

bounded by regular surfaces, undergoing piecewise differentiable paths, they compute the envelope by sweeping a set of *critical curves* and edges on the boundary faces of the generator. Considering a family of surfaces expressed by $e=r(u\ (v, t), v, t)$, any point on the envelope satisfies the property

$$\left|\frac{\partial r}{\partial u}\frac{\partial r}{\partial v}\frac{\partial r}{\partial t}\right|=0 \qquad (6)$$

This can be used to find the family of critical curves whose motion defines the boundary of the swept volume. They have implemented their theory for spherical and cylindrical generators and apply it to the graphical verification of NC tool paths.

Hu and Ling[15] also employ envelope theory in computing the volumes swept by generators modelled with the natural quadrics (planes, spheres and circular cylinders and cones). They use the instantaneous screw axis to describe the motion of the generator.

Martin and Stephenson[16] present a theoretical foundation for computing swept volumes of a three-dimensional object as it moves along an arbitrary path, also by using envelope theory. Their theory extends to all surfaces defined by an implicit equation of the form $f(x, y, z)=0$. When this surface moves, it defines a family of surfaces (parametrized by t), $F(x, y, z, t)=0$. The envelope must simultaneously satisfy this equation and $\partial F(x,y,z,t)/\partial t=0$. They explain that simply by eliminating t one can (in principle) obtain an implicit equation for the envelope surface, of the form $e(x, y, z)=0$. They additionally explain how one might combine the envelopes generated by sweeping the individual surfaces of a solid body to form a solid model of the swept volume.

They present only the theory and recognize that there are many degenerate cases which must be handled. Also, they explain that methods of computer algebra may or may not be able to eliminate t and produce an implicit equation that is useful for representing the swept volume. In addition, their technique will often require the employment of exponential algorithms in computer algebra for computing even simple sweeps of simple surfaces. As they note, further work is needed to determine what types of surfaces and motions can be realistically computed using their proposed methods.

Weld and Leu[17] set forth some theory for computing the volume swept by a moving polyhedron. They prove in general terms that the volume swept by a compact *n-manifold* as it moves through $\mathbf{R}^n$ is equal to the union of the volumes swept by its $(n-1)$-dimensional boundary elements, combined with one location of the *n-manifold* during the sweep. Notably, when $n=3$, this implies that the volume swept by a moving polyhedron A can be computed by unioning the volumes swept by the boundary elements of the polyhedron (i.e. its faces) together with one location of the polyhedron itself during the sweep. (If it is possible to find two times i and j during the sweep where $A_i \cap A_j=\varnothing$, they show that the addition of a copy of the polyhedron is unnecessary.) Thus an important step in computing the volume swept by a moving polyhedron is to compute the volume swept by each of its moving polygonal faces.

To solve this subproblem, they observe that the volume swept by a polygon moving in 3-space is bounded by the *ruled surfaces* swept by the moving edges of the polygon, copies of the polygon at its initial and final positions, and sometimes by the volume swept by the *interior* points of the polygon. They observed that these points from a polygon's interior appear on the boundary of the swept volume only if there are successive instances of the polygon at times t and $t+\varepsilon$ which intersect each other. The moving polygon's plane is a one-parameter family of planes—this parameter is time. The envelope of a one-parameter family of planes is known to be a developable surface. They found that in these cases of successive intersections, portions of this developable surface may appear on the boundary of the swept volume. See, for example, Figure 1. They also present point-wise tests for determining which points of the ruled surface segments and the developable surfaces are in fact on the boundary of the swept volume.

We have found that their condition for determining if portions of this developable surface can appear on the boundary of the swept volume is not a necessary condition. We shall examine this further in our discussion of what we call 'sliding motion' later. Weld and Leu's work also stops short of defining the

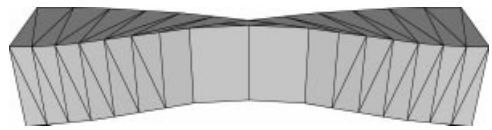

Figure 1. This figure shows the volume swept by a square as it moves left to right, rotating as it translates. Notice that as it passes the middle of its trajectory it can be seen to be instantaneously 'sliding' across the page. In this vicinity there are clearly points on the boundary of the swept volume which came from the interior of the polygon.

actual boundary representation for the swept volumes. They explain that this would require computing all the surface–surface intersections between the ruled and developable surface segments to form a set of surface patches, a subset of which would be the boundary of the swept volume. Their point-wise tests, while of theoretical interest, do not seem to be of practical use for this purpose.

Sambandan[18] and Sambandan and Wang[19] employ envelope theory by directly solving the envelope equations for specific sweep generators—specifically, those used for NC cutting tools (spheres, hemispheres and fillet surfaces). Since the envelopes are solvable for simple geometries such as these, their approach makes sense for NC verification. However, they too make a simplification based on a 'successive intersection' test that, like the Weld and Leu[17] test, is not a necessary condition.

Other Representations

Other researchers have considered implicit or functional representations of solids and the volumes swept by them, such as Schroeder *et al.*[7] In this work a function $f(p)=0$ defines the surface of a solid to be moved, where $f(\ )$ defines the distance from a point p to the surface of the object in $\mathbf{R}^3$. The swept volume is computed by sampling the transformed implicit model as it is swept along the trajectory (using interpolation for anti-aliasing), producing an array of distance values making up the workspace volume. The swept surface is extracted via the marching cubes surface extraction algorithm[20] and reduced using a decimation algorithm.[21] A benefit of their method is its automatic handling of self-intersections. However, their method does not automatically handle the extraction of multiple surfaces on the boundary of the swept volume.

Another approach using implicit models is taken by Sourin and Pasko.[22] They consider unions of functionally defined moving solids, defined by a function $f(x, y, z, t)\geqslant 0$, and attempt to compute a functional representation of the swept solid, $F\ (x, y, z)\geq 0$. They propose two different approaches. The first uses unions of the object at discrete time intervals $\delta - t$, using R-unions[23] for the analytical computations of the set-theoretic union operation. To this they add a blending function to avoid aliasing artefacts.

Secondly, they consider methods of directly computing a functional representation of the envelope of the moving solid. They present a numerical algorithm for finding the envelope defining function and use this for computing the swept volume. Interestingly, owing to the generality of their method, theirs is the only method seen which can handle arbitrary motions of *non-rigid* bodies, i.e. objects which deform as they move. They claim that their method is the first to handle, within a single algorithm, the sweeps of arbitrary variable-shape solids with procedural defining functions, the creation of solids via CSG-like schemes, and the sweeps of arbitrarily complex objects over arbitrary motions while handling self-intersections. We have also not found any other method which handles all these problems. They also point out the primary shortcoming of their method, which is its speed, a common problem in all sweeping methods.

Blackmore *et al.*[6] combine implicit representations with envelope theory and present the derivation of the sweep envelope differential equation (SEDE) for computing swept volumes along with a description of its implementation. Essentially, they compute a discrete approximation to the *grazing set* by triangulating the surface of the object, identifying an initial set of grazing points which satisfy the SEDE, and discretely stepping these points along their approximate trajectories. This set is trimmed and triangulated (via linear interpolation), yielding an approximation of the swept volume. Their method requires representing the generator by a continuous function $f(x, y, z)=0$ (approximating it if necessary). Their implementation demonstrates that this technique works well for smooth generators, although they did not describe how their triangulation handles self-intersecting swept surfaces.

Menon *et al.*[24] employ a ray representation—a set of ray segments classified as 'in', 'on' or 'out' of the object being represented—to handle a number of geometric operations. Their highly parallel ray-casting engine can efficiently compute approximate Boolean operations on ray-reps of objects, including Booleans. They then compute swept volumes by a set of discrete unions of the generator at several positions along its trajectory.

Why Find Another Method

The research described above can be divided into three categories. The first category contains research that discusses theoretical foundations and properties of general swept volumes. The second category contains those approaches which restrict the classes of motion permitted, the class of shapes being swept, or both. These produce useful results in specific domains, as is

the case in References 4 and 19. The work of Blackmore *et al.*[6] also falls into this category. Their excellent extensions of envelope theory—and their successful implementation of those ideas—are very useful in domains where the generator object has a continuous surface, such as NC cutting tools which can often be approximated by spheres or cylinders. In the third category are those methods which can handle arbitrary shapes but require input and/or yield outputs in inconvenient forms. Each of the above sets of constraints is appropriate in particular domains. However, in many applications it is important to allow arbitrary polyhedral models to move through arbitrary trajectories while maintaining a typical boundary representation of the input and output. The algorithm presented below accomplishes this task.

Approximating the Swept Volume

As Weld and Leu[17] showed, the boundary of the swept volume may contain portions of polygonal surfaces, ruled surfaces (from the moving edges) and developable surfaces (the envelope of the moving polygons' planes). An *exact* computation of the volume swept by a moving polygon would therefore require computing an intersection graph of these surfaces. Once done, the intersection graph can, in theory, be traversed to find the boundary of the swept volume. As Martin and Stephenson[16] showed, simply employing computer algebra techniques to find equations for the envelopes may not be tractable. Further, even if a representation for these surfaces can easily be found for an arbitrarily moving polygon, they all need to be intersected with one another. This process itself is an expensive proposition with many robustness problems, and may not even be possible, depending on the types of motion undergone. This, combined with the fact that the visibility computation employed as part of the sensor planning process requires polyhedral inputs, led us to focus on methods of computing polyhedral approximations to these volumes.

Our basic approach has been to approximate the ruled surfaces with triangulated meshes and to approximate the developable surfaces with copies of the object's polygons stepped through their trajectories. We combine these meshes and polygons and compute their total arrangement, and finally traverse the outer cell in this arrangement to find the boundary representation of the swept volume. In this section we present the details of an algorithm which follows this approach.

Approximating the Ruled Surface

The surface swept by a moving edge is a ruled surface given by the equation

$$r(t, s) = p(t) + s.v(t) \tag{7}$$

where the position and orientation of the ruling line segment are parametrized by t and the position along the line segment is given by s. At parameter values t_0 and t_1 the rulings can be drawn and the corresponding end points can be connected by line segments. Another line can then be drawn between one opposite pair of end points, creating two triangles approximating the surface. See Figure 2. When $t = t_0$, the ruling line $\overline{AD}$ is exactly on the surface. When $t = t_1$, line $\overline{BC}$ is exactly on the surface. As drawn, A and B are points with the same s parameter value (say s_0) and C and D have a second parameter value (s_1). Connecting corresponding vertices A and B with a line segment, corresponding vertices C and D with a segment and then opposite vertices D and B with another segment yields two triangles ΔABD and ΔDBC. These triangles approximate the ruled surface on the parametric interval $([t_0, t_1], [s_0, s_1])$.

In this fashion, each edge of each polygon of the polyhedron to be swept can be stepped through its trajectory, creating a triangulated mesh approximating the ruled surface swept by each edge. Figure 3 shows an example of one edge being stepped through its trajectory, and the resulting triangle mesh approximating the ruled surface. In general, however, all edges need not be considered at all steps.

In particular, the ruled surface swept by a moving edge during a given time interval can only contribute to the boundary of the swept volume if the edge's motion is towards the 'outside' of the polygons

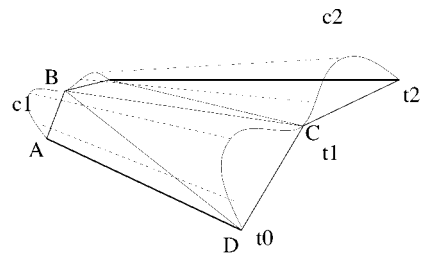

Figure 2. Approximating a patch of a ruled surface.

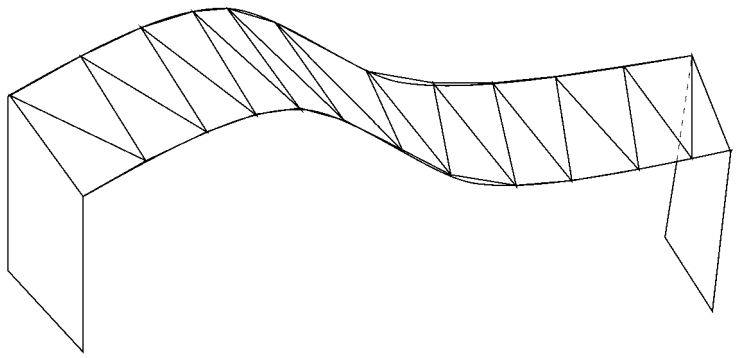

Figure 3. The edge is stepped through its trajectory. The resulting triangulated mesh approximates the ruled surface.

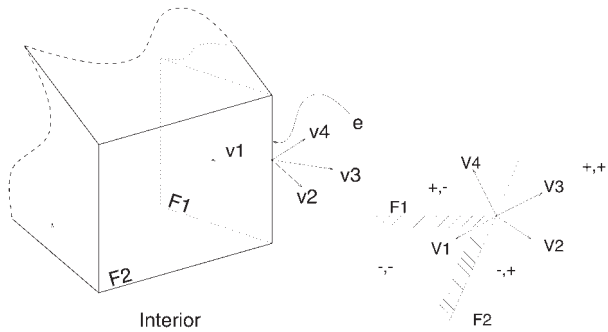

Figure 4. Determining if an edge is needed at a given moment. Faces F_1 *and* F_2 *are incident at edge e. If e is, during the given step, sweeping locally 'into' the object, then the ruled surface generated by that edge during that step will not be on the boundary of the swept volume. The figure on the right shows a cutaway view looking down on edge e.*

incident at that edge and if the motion of the edge is not subsumed by the motion of the faces. For example, see Figure 4, which shows faces F_1 and F_2 adjacent to one another at an edge e. Each face separates the world into two half-spaces, i.e. the front and back sides of its plane. The pair of faces F_1 and F_2 divides the world into four quadrants. Let us label the 'outside' or 'front' of each plane the positive half-space, and the 'inside' or 'back' of each plane the negative half-space. The four quadrants can now be labelled according to these signs.

First let us examine convex edges* considering only their instantaneous translational motion for the moment. When an edge is moving into the $(-,-)$ quadrant, then the edge's motion is to the interior of the polyhedron. Therefore the ruled surface swept by this edge need not be considered. When an edge's motion is to the $(+,+)$ quadrant, then the edge's motion will be subsumed by that of the adjacent faces. If, however, the edge moves into either the $(+,-)$ or $(-,+)$ quadrant, then the surface swept by the edge may appear on the boundary and therefore needs to be considered. When rotations as well as translations are permitted, then it is possible that motions into the $(+,+)$ quadrant may cause the surface to appear on the outside. Therefore, in the general case, we can only exclude edges which are, at a given step, locally moving into the $(-,-)$ quadrant. As we are discretely stepping each edge, we can compute the motion of the edge by computing the motion of its end points between times t_i and t_{i+1}. If these motion vectors are into the $(-,-)$ quadrant, then the triangulated surfaces need not be computed between these two times. This test must be performed looking at the quadrants formed by the faces at both times t_i and t_{i+1}.

Reflex edges, however, can never create ruled surfaces which contribute to the boundary of the swept volume, because these ruled surfaces are always subsumed by the motion of the incident faces. See Figure 5. The motions of these edges are either towards

*That is, edges for which the incident faces meet such that the interior dihedral angle is less than 180°. When this angle is greater than 180°, the edge is called a reflex edge.

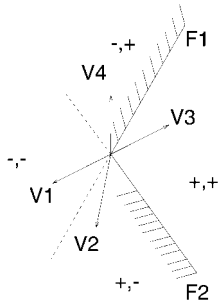

Figure 5. If edge e is not a convex edge, the ruled surface generated by its motion is always subsumed by the motion of the incident faces.

the interior of the polyhedron or towards the $(+,+)$ quadrant, and owing to the reflex angle, the motion of the edge is always subsumed by that of the incident faces.

Therefore it is only necessary to consider certain convex edges of the polyhedron. Interestingly, the tests applied to determine which edges are necessary here in the sampled domain seem analogous to the tests used in Reference 6 to determine grazing sets in the continuous domain.

Approximating the Envelope Surface

In addition, as we said, there are times when the envelope of the moving polygon's interior appears on the boundary of the swept volume as well. This can happen, as Weld and Leu[17] found, when successive copies of the polygon instantaneously intersect one another, as we illustrated in Figure 1. However, it is not necessary for such a successive intersection to take place.

A condition which is necessary is that there must be some point of the polygon whose velocity in the direction normal to the plane of the polygon is zero. That is to say, point r, an interior point of the polygon, will appear on the boundary of the swept volume only if there exists a time t at which $v(r, t)\cdot\vec{n}_t = 0$ (where $v(r, t)$ is the instantaneous velocity of point r at time t and $\vec{n}_t$ is the polygon's normal at time t) (R. T. Farouki, personal communication, 23 May 1994). Note that this is not a sufficient condition, as there are points which will meet this criterion but not appear on the boundary of the swept volume.

That this condition is necessary can be shown via a proof by contradiction. Assume that there is a point r on the interior of the polygon which appears on the boundary of the swept volume at time t. Further assume that it does not meet the condition stated above, i.e. assume that $v(r, t)\cdot\vec{n}_t \neq 0$. That means that at time t, the moment at which this interior point of the polygon appears on the boundary of the swept volume, its velocity is non-zero in the direction of the polygon's normal. Let P_t be the polygon positioned as it would be at time t and let r_t be the position of point r at time t. Since its velocity at t is non-zero in the direction of n_t, there exists some small ε for which $r_{t-\varepsilon}$ is on one side of polygon P_t and $r_{t+\varepsilon}$ is on the other side of P_t.

Furthermore, since r is an interior point of P, then there is a neighbourhood of points around r and P—a small disc in the plane of P—which are all on one side of P_t at $t-\varepsilon$ and on the other side of P_t at $t+\varepsilon$. All these points together make up a neighbourhood in three dimension around r_t, and this entire neighbourhood is clearly part of the swept volume. Since r_t is interior to this neighbourhood, it is interior to the swept volume and cannot be on the boundary of the swept volume. Therefore, if an interior point r of P appears on the boundary of the swept volume, there must be some time t at which $v(r, t)\cdot\vec{n}_t$ necessarily must be zero.

When there exists such a point r at time t, we say that the polygon is undergoing 'sliding' motion, because the polygon is sliding in its own plane, at least locally. Refer again to Figure 1. In this figure a square is translating from left to right, rotating as it goes. As it passes the middle of its trajectory, it can be seen to be instantaneously 'sliding' in front of us. In this vicinity there are clearly points on the boundary of the swept volume which came from the interior of the polygon. This sort of motion meets Weld and Leu's[17] criterion of 'successive intersections' as well.

However, Figure 6 shows a square that is translating in a semicircular trajectory *without* rotation. The plane of the semicircle is perpendicular to the plane of the square. That is, the trajectory goes 'into' the page. In this example, when the square is at the mid-point of its trajectory, there are clearly points on the boundary of the swept volume which came from the polygon's interior. For example, the centre of the square is on the back boundary of the swept volume. There can be no

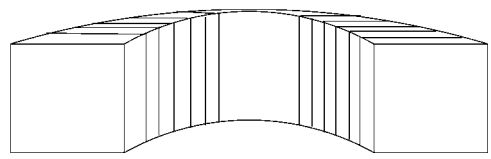

Figure 6. Another example of sliding motion.

successive intersections, since all successive copies of the polygon are parallel to one another. However, at that same farthest point in the trajectory the instantaneous velocity of the polygon is zero in the direction of the surface normal, thereby meeting our criterion. Again, the face is, at that moment, 'sliding' in front of us.

When sliding motion takes place, we need to include an approximation to the envelope of the family of planes given by the moving polygon. We have seen that sliding motion takes place when some point on the polygon has zero velocity in the polygon's normal direction. Looking at the polygon's motion at the same discrete steps as we looked at each edge above, we can determine if sliding motion took place between t and $t+\Delta t$. The following algorithm will compute $\mathscr{S}$. A set of faces which will be useful for approximating the developable surface formed by the interior of a moving polygon p when that polygon undergoes sliding motion.

Sliding Motion Test

1. Compute the motion of each vertex of p from t_i to t_{i+1} relative to p's normal at time t_i. The vertex will be moving forward, backward or in the plane of p. For each vertex v_j let $\sigma_{t_i}(v_j)$ be $+1$ (forward), -1 (backward) or 0 (in the plane), depending on its motion at t_i.
2. If any face has vertices moving in different directions, i.e. if $\sigma(v_j) \neq \sigma(v_k)$ for any v_j and v_k of p (such as if the face is rotating about an axis in its plane), or if there are any vertices v_l for which $\sigma(v_l) = 0$, the face is undergoing sliding motion between t_i and t_{i+1}. Add copies of the face at both t_i and t_{i+1} to $\mathscr{S}$.
3. If for any vertex v_j of p, $\sigma_{t_{i-1}}(v_j) \neq \sigma_{t_i}(v_j)$, i.e. if any vertex has changed direction from the previous interval (relative to the face's plane, that is), the face is undergoing sliding motion at t_i. Add a copy of the face at t_i to $\mathscr{S}$.

This algorithm is run at every intermediate step t_i *between* the start time and the end time (i.e. not at the end points of motion themselves).

For an example, see Figure 7. In this figure, notice that polygon p is pivoting about the dotted axis. Vertices A and B move into the page to corresponding vertices A′ and B′, while vertices C and D move in the opposite direction to corresponding vertices C′ and D′. Therefore this face undergoes sliding motion between times t_0 and t_1, according to step 2 above, and p_{t_0} and p_{t_1} need to be added to the set $\mathscr{S}$.

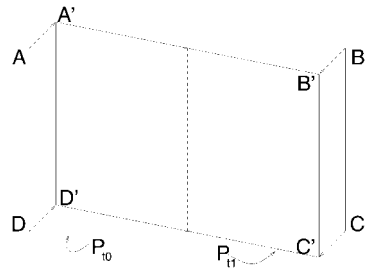

Figure 7. Sliding motion: vertices A and D are moving in one direction relative to the plane of P_{t_0}, while vertices B and C are moving in the other direction.

In Figure 8, polygon p changes direction at time t_i. That is, immediately prior to t_i, all its vertices were moving into the page, and immediately after t_i, all its vertices were moving out of the page. Thus, according to step 3 above, p_{t_i} needs to be added to $\mathscr{S}$.

Combining the Surfaces

The set of polygons, $\mathscr{S}$, produced with the above algorithm, plus copies of the moving polygon at the start and end points of motion, plus the set of triangles generated to approximate the ruled surfaces from each edge, yield a new set of polygons, $\mathscr{F}$. $\mathscr{F}$ is a superset of the boundary of our approximation to the volume swept by a moving polygon. Specifically, each member of $\mathscr{F}$ is completely interior to, completely on the boundary of, or partially interior to and partially on the boundary of the swept volume. That is, by construction, no member of $\mathscr{F}$ is partially or completely exterior to the swept volume.

If there were never any self-intersections in the swept surfaces, one could simply stitch the elements of the set $\mathscr{F}$ along their edges and traverse the outer boundary of this set. To handle cases in which self-

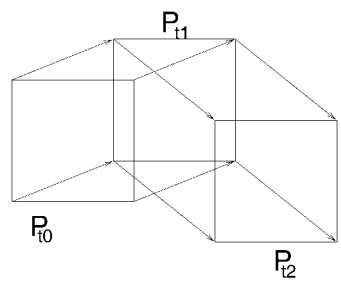

Figure 8. Sliding motion due to a change in direction. From time t_0 to t_1, all vertices of P are moving in one direction relative to its plane. Between times t_1 and t_2, they are moving in the other direction.

intersections do occur, it is important to compute all intersections among the faces of $\mathscr{F}$—that is, compute the arrangement of these polygons in space yielding a collection of cells (see e.g. Reference 25).

Most of these cells will be in the swept volume, but some of these cells may be voids within the swept volume. Clearly, an object which itself contains voids can produce a swept volume which contains voids. For example, any hollow object which simply translates a short distance in a straight line can produce a swept volume with at least one interior void. In addition, there are motions of voidless objects which will produce voids in the swept volume.

Because voids are unimportant for our own sensor planning work, the current implementation does not find voids. Instead, we determine only the outer boundary for the swept volume. That is, we compute the arrangement of all polygons in the set $\mathscr{F}$. Then we traverse the outer boundary of this arrangement via a gift-wrapping algorithm. This yields the outer boundary of the swept volume, having all self-intersections taken into consideration.

Implementation

We have implemented the following algorithm for generating the outer boundary of the swept volume $\mathscr{S}(P, T)$ for a polyhedron P and a trajectory T.

Polyhedral Sweep Algorithm

1. Step each edge of P through the trajectory T using any step size Δt. Connect adjacent copies of each edge by forming triangles, as shown in Figure 2. Collect these triangles in a set called $\mathscr{F}$.
2. For each polygon p in P, do the following.
   (a) Add a copy of p_i at its initial and final positions to $\mathscr{F}$.
   (b) Run the *sliding motion test* (above) on p_i, using the same step size Δt. Add all the faces placed in S by this algorithm to $\mathscr{F}$.
3. Set $\mathscr{F}$ is now a superset of the boundary of the actual volume swept. Compute the set $\mathscr{A}$ = Arrangement ($\mathscr{F}$).
4. Traverse the boundary of the infinite cell, i.e. the outermost boundary of $\mathscr{A}$, and regularize it (discarding all faces not needed to define the this outermost boundary). The resulting volume is $\mathscr{S}(P, T)$.

As we have discussed, this algorithm will compute the outer boundary of the swept volume. The algorithm was implemented in C++ on a Sparc-20 and used the ACIS geometric modelling system for its boundary representation, Boolean operations, boundary traversal algorithm, and save and load of the generator and computed swept volume (to and from ACIS .SAT format). In the actual implementation the arrangement $\mathscr{A}$ was built incrementally out of subsets of $\mathscr{F}$. For example, as an edge was stepped along its trajectory, the triangulated mesh arising from that edge was grouped together and its arrangement was computed independently of the rest of the faces. Later, all these subsets were combined and their total arrangement was computed. This yielded gains in efficiency and robustness over brute-force methods. The arrangements were computed using sequences of 'union' operations, although more elegant means could be used[25] for greater gains in efficiency and robustness.

The algorithm has been run on a number of examples, of which a few are selected for inclusion here. Figure 9 shows a rectangular prism with dimensions $5 \times 6 \times 7$, turned so as to sit on one vertex. This prism is swept in the helical trajectory given by

$$H(t) = \begin{bmatrix} \mathrm{Rot}(z, 0.1t\pi) & T(0, 8, t/2) \\ 0 \ldots 0 & 1 \end{bmatrix} \qquad (8)$$

(where Rot (z, θ) implies a rotation about the z-axis by θ, and T (x, y, z) implies a translation by (x, y, z)), with t stepped from 0 to 1 in steps of 0·1. The volume generated is shown in Figure 10. This example does not exhibit sliding motion, although the following examples all do.

The object shown in Figure 11 is swept through a spiralling trajectory along the y-axis, given by a translation of $(\sin(2\pi t/10), t, 0)$ followed by a rotation of $\mathrm{Rot}(y, t\pi/20)$, and the results are displayed in Figure 12. The same object rotated 90° about the x-axis and swept through the helical trajectory of equation (8) is shown in Figure 13. Figure 14 displays the result of applying the swept volume algorithm to a robotics problem, in which a model of the Barrett dextrous robot hand is shown sweeping from a partially spread position to a partially closed position.

Figure 9. Rectangular prism to be swept.

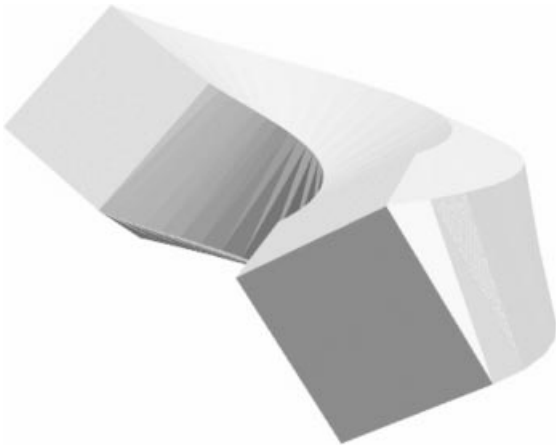

Figure 10. Volume generated from helical sweep.

Figure 11. Object to be swept.

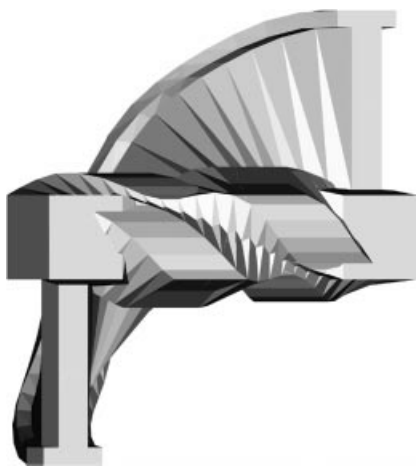

Figure 12. The computed swept volume.

The hand started with the palm at a spread angle of 135° and closed to a 0° spread, while the fingers went from fully open (0° to 30° closed. Figure 15 shows the volume swept by a Puma 560 robot with a metal stylus attached to its end-effector moving in a 'V-shaped' trajectory.

Finally, Figure 16 shows a sweep of the DLR robotic hand. To start, the hand was placed in the configuration shown in the left portion of the figure, and the links were commanded to move to the positions shown in the right portion of the figure. The two images in the centre of the figure show the volume swept by the links executing this motion, assuming that the joints all move at equal velocities.

Handling Voids

While voids inside the swept volume are not important for certain applications such as our sensor planning research (a camera placed within a void will be unable to see anything outside of the void), a general-purpose swept volume algorithm should be able to find them. There is a conceptually simple extension to our algorithm which will find voids in the result. Essentially, one must compute each cell of $\mathscr{A}$ rather than simply traverse the outer boundary of the entire arrangement. Each cell will then need to be classified to determine if it is part of the volume or a void within the volume. The union of all the cells which pass the classification test then comprises the swept volume. In practice, however, it is likely to be a bit more complicated.

Every face, as it is generated, can be labelled so as to remember the generator element which gave rise to it. In addition, the side of the face which corresponds to the inside of the swept volume, locally, can be remembered. While this local information will be useful (and probably necessary), it is not likely to be sufficient for classifying the cell. What might appear to be a void, locally, may be contained within the sweep of another element. Therefore a combination of local and global methods (such as ray casting) will probably be required in order to determine a *nesting* of the cells. This nesting should be very useful in assisting the identification of voids. These techniques may be computationally expensive. Therefore, if an algorithm can determine *a priori* that a void will not exist in the result, the most efficient method would be to traverse the outer boundary rather than to identify and classify each of the cells which arise in the arrangement. It is

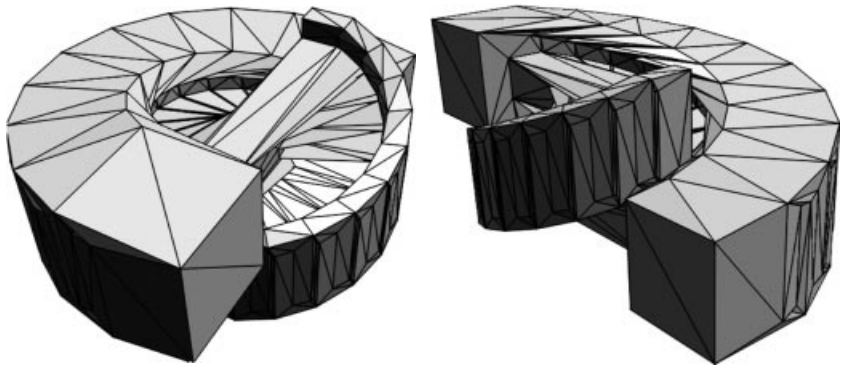

Figure 13. The same object rotated 90° about the x-axis and spiralling upwards. Shown in two views, with all edges highlighted.

Figure 14. Computed swept volume of a Barrett hand, closing.

hoped that these ideas will prove to be useful guides for future researchers in swept volume computation.

Performance Issues

As was observed in Reference 22, all methods of computing swept volumes are slow. This one is no different. Construction of the set $\mathcal{F}$ is not at all expensive (trivially, $O(nm+fm)$, where n is the number of edges, m is the number of steps and f is the number of faces, since each edge creates $O(m)$ triangles, each found in constant time, and each face may create $O(m)$ copies of itself, each of which can be found in constant time). Similarly, the traversal of the arranged faces is not expensive (linear in the number of faces in the arrangement). However, the computation of the arrangement itself can be very time-consuming. As described above, we perform the arrangement computation by sequences of union operations on sets of faces as they are generated. Since the proprietary union of a commercial CAD package was used, we cannot accurately analyse the complexity of this technique. However, we can discuss the theoretical bounds for arrangement computations in general.

For example, a full arrangement of n triangular faces can be computed in $O(n^3)$. Better, an algorithm for

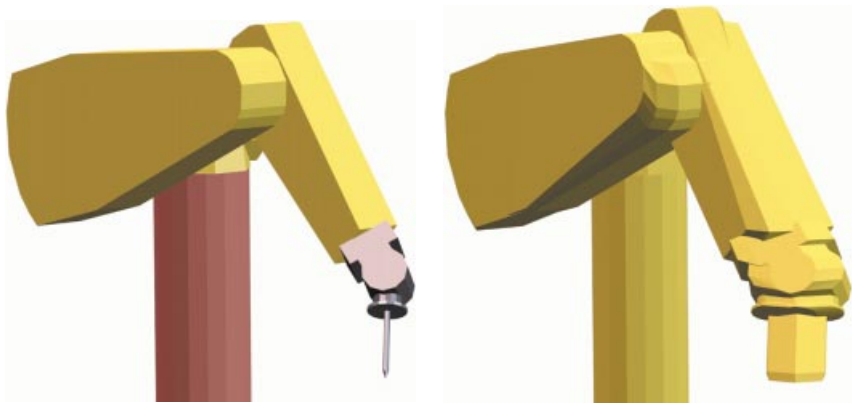

Figure 15. Puma model and computed swept volume.

Figure 16. Left: initial configuration of the DLR hand. Centre: front and side views of the swept motion of the links. Right: final configuration of the hand.

computing a single cell in randomized expected time $O(n7/3+\lambda)$, with the constant of proportionality depending on λ, is presented in Reference 26. This bound (and algorithm) is applicable to our problem, since the traversal of the outer boundary yields the one and only cell in which we are interested. While most of our faces are triangular, copies of the moving polygons are not, although their algorithm could easily be applied by first triangulating these polygons.

Despite the cost of the arrangement computations, one can make a speed/accuracy trade-off by adjusting the step size. This might be useful, for example, in visualization or animation applications, where a quicker, less accurate method can be used initially and more precision can be obtained, at the price of speed, when necessary.

One significant problem we have encountered is that both commercial and research geometric engines are often not robust enough to handle the arrangement computations discussed above (owing to floating-point error and related issues) if the step size is too small or the object is too complex. Robustness is a well-known problem and much studied in computational geometry. A good deal of research has gone into issues of handling degeneracies, using more precise arithmetic, and related issues of robustness in geometric computing, summarized in Reference 27. As steps are taken to improve the robustness of the underlying geometric engines, this problem can be minimized in this and many other applications.

Interestingly, in the cases for which an arrangement cannot be computed, we are able to take the set of polygons, $\mathcal{F}$, and graphically render them, displaying what the result should look like; therefore this method can be used for graphically displaying even those swept volumes that it cannot compute. There are in fact a number of applications where graphical displays of swept volumes are sufficient.

Conclusion

This paper has presented a new method for the computation of swept volumes. While a good deal of research into the nature and computation of swept volumes has been done over the years, much of the work we have seen suffers from one or more limitations, such as a restriction on the class of motion, the shape of the objects moving or the way in which the objects must be represented. These limitations prevent much of this other research from being used in practical applications. The method presented in this paper has the benefit of accepting polyhedral boundary representations as input and producing polyhedral boundary representations as output, with no restrictions on the class of motion involved (subject to the robustness of the underlying geometric operators).

As we showed via the examples, this algorithm is robust enough to produce valid results in fairly complicated, useful situations. Further, in applications where a graphical representation of the swept volume is all that is required, the methods presented can be used to quickly generate a set of faces to be rendered via standard Z-buffering techniques.

However, there is more work which can be done. The swept volume computation is, as was discussed above, very expensive. Despite this, the primary limitation of the swept volume computation is not its speed, but its robustness. It is one thing to have an algorithm which takes a long time to produce a useful result; it is another thing to have a specific implementation which takes a long time to *fail*. As we mentioned, there is ongoing research in the computational geometry community to help minimize robustness problems in applications such as this.

ACKNOWLEDGEMENTS

This work was supported in part by NSF grants CDA-96-25374 and IRI-93-11877. We would also like to thank Professor Gerd Hirzinger and Dr Max Fischer from the German Aerospace Center (DLR) for providing us with models of their robotic hand. Finally, we would like to thank Andrew Miller for generating the images of the DLR hand.

References

1. Hui KC, Tan ST. Mould design with sweep operations—a heuristic search approach. *Computer Aided Design* 1992; **24**: 81–91.
2. Korein J. *A Geometric Investigation of Reach*. MIT Press: Cambridge, MA, 1985.
3. Cameron SA. Modelling solids in motion. *PhD Thesis*, Department of Computer Science, University of Edinburgh, 1984.
4. Wang WP, Wang KK. Geometric modeling for swept volume of moving solids. *IEEE Computer Graphics and Applications* 1986; **6**(12): 8–17.
5. Sungurtekin UA, Voelcker HB. Graphical simulation and automatic verification of NC machining programs. In *Proceedings 1986 IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 1986. IEEE Computer Society Press: Washington, DC, 1986; 156–165.
6. Blackmore D, Leu MC, Want LP. The sweep-envelope differential equation algorithm and its application to NC machining verification. *Computer Aided Design* 1997; **29**: 629–637.
7. Schroeder WJ, Lorensen WE, Linthicum S. Implicit modeling of swept surfaces and volumes. In *Proceedings IEEE Visualization Conference*, Washington, DC, October 1994. IEEE Computer Society Press: Los Alamitos, CA, 1994.
8. Jayaram S, Connacher HI, Lyons KW. Virtual assembly using virtual reality techniques. *Computer Aided Design* 1997; **29**: 575–584.
9. Abrams S. Sensor planning in an active robot work-cell. *PhD Thesis*, Columbia University, New York, 1997.
10. Abrams S, Allen PK, Tarabanis KA. Dynamic sensor planning. In *Proceedings 1993 IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993. IEEE Computer Society Press: Los Alamitos, CA, 1993.
11. Abrams S, Allen PK, Tarabanis KA. Computing camera viewpoints in an active robot work cell. *International Journal of Robotics Research* 1999; **18**: 267–285.
12. Abrams S, Allen PK. Computing swept volumes for sensor planning tasks. In *Proceedings DARPA 1994 Image Understanding Workshop*, Monterey, CA, November 1994. Morgan Kaufmann: San Francisco, CA, pp. 1159–1166.
13. Kaul A. Computing Minkowski sums. *PhD Thesis*, Department of Mechanical Engineering, Columbia University, New York, 1993.
14. Hui KC. Solid sweeping in image space—application in NC simulation. *The Visual Computer* 1994; **1**: 306–316.
15. Hu Z-J, Ling Z-K. Swept volumes generated by the natural quadric surfaces. *Computers and Graphics* 1996; **20**: 263–274.
16. Martin RR, Stephenson PC. Sweeping of three-dimensional objects. *Computer Aided Design* 1990; **22**: 223–234.
17. Weld JD, Leu MC. Geometric representation of swept volumes with application to polyhedral objects. *International Journal of Robotics Research* 1990; **9**(5): 105–117.
18. Sambandan K. Graphic simulation and verification of five-axis NC-machining. *Master's Thesis*, Cornell University, Ithaca, NY, 1988.
19. Sambandan K, Wang KK. Five-axis swept volumes for graphic NC simulation and verification. In *Proceedings 15th ASME Design Automation Conference*. Montreal, September 1989, Vol. 19–1. Ravani B (ed). ASME: New York, 1989; 143–150.
20. Lorensen WE, Cline HE. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics* 1987; **21**(4): 163–169.
21. Schroeder WJ, Zarge JA, Lorensen WE. Decimation of triangle meshes. *Computer Graphics* 1992; **26**(2): 65–70.
22. Sourin A, Pasko A. Function representation for sweeping by a moving solid. In *Proceedings Third Symposium on Solid Modeling and Applications*, Salt Lake City, UT, May 1995. IEEE Computer Society, ACM Press: New York, 1995; 383–391.
23. Shapiro V. Real functions for representation of rigid solids. *Computer Aided Geometric Design.* 1994; **11**: 153–175.
24. Menon J, Marisa RJ, Zagajac J. More powerful solid modeling through ray representations. *IEEE Computer Graphics and Applications* 1994; **14**(3): 22–35.
25. Edelsbrunner H. *Algorithms in Combinatorial Geometry*. Springer: Berlin, 1987.
26. Aronov B, Sharir M. Triangles in space or building (and analyzing) castles in the air. *Combinatorica* 1990; **10**: 137–173.
27. Fortune S. Robustness issues in geometric algorithms. In *Proceedings 1996 Workshop on Applied Computational Geometry*, Philadelphia, PA, May 1996: pp. 20–23.

Authors' biographies:

Steven Abrams is the manager of the Computer Music Center in the Department of Mathematical Sciences at IBM Research. Prior to becoming immersed in the field of computer music, Dr. Abrams developed 2-D and 3-D geometric processing algorithms and system architecture for a rapid prototyping system. He then joined Stratasys, Inc., where he helped to commercialize that rapid prototyping technology. That technology is currently shipping in Stratasys' ''Genisys'' product. He studied at Columbia University where he earned the B.S., M.S., M.Phil., and Ph.D. degrees. His Ph.D. thesis was on sensor planning for robots in an active environment, and focused on multidimensional modeling and manipulation of computer vision constraints and the computation of 3-D swept volumes.

While still interested in 3-D modeling and visualization, his primary research focus is on computer music issues – particularly in developing software tools that utilize higher-level abstractions of important musical concepts in ways that enhance the composer's ability to create music.

Peter K. Allen is associate professor of Computer Science at Columbia University. He received the A.B. degree from Brown University in Mathematics–Economics, the M.S. in Computer Science from the University of Oregon and the Ph.D. in Computer Science from the University of Pennsylvania, where he was the recipient of the CBS Foundation Fellowship, Army Research Office fellowship and the Rubinoff Award for innovative uses of computers. His current research interests include real-time computer vision, dextrous robotic hands, 3-D modeling and sensor planning. In recognition of his work, Professor Allen has been named a Presidential Young Investigator by the National Science Foundation.